НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 519.688

«До захисту допущено»
Завідувач кафедри СПСКС
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
"___" _____ 2018 р.

Магістерська дисертація

на здобуття ступеня магістра

з напряму підготовки 123 Комп'ютерна інженерія
Спеціалізовані комп'ютерні системи

на тему: ПРОГРАМНІ ЗАСОБИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ РОБОТИ СУБД MONGODB

Виконав: студент II курсу, групи КВ-73мп
(шифр групи)

Даценко Сергій Олександрович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник: доц.каф.СПіСКС, к.т.н., доцент Петрашенко А.В. _ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент: _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем
Рівень вищої освіти – другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Спеціалізовані комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
"___" _____ 2018 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Даценку Сергію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Програмні засоби підвищення продуктивності роботи СУБД MongoDB»

науковий керівник дисертації доц.каф.СПСКС, к.т.н., доц. Петрашенко А.В.

затверджені наказом по університету від 30 жовтня 2018 р. №4030-с

2. Термін подання студентом дисертації: 7 грудня 2018 р.

3. Об'єкт дослідження: розподіл даних між серверами в кластері MongoDB.

4. Предмет дослідження: розробка програмних засобів для підвищення продуктивності роботи СУБД MongoDB, які призначені для налаштування розподілу даних відповідно до допустимих відсоткових навантажень вузлів кластера, враховуючи вплив «джамбо-чанків».

5. Перелік завдань, які потрібно розробити:

- розглянути та проаналізувати існуючі підходи розподілу даних;
- запропонувати новий підхід розподілу даних;

- розробити програмні засоби для реалізації запропонованого підходу;
- протестувати роботу запропонованого підходу, зробити висновки.

6. Перелік ілюстративного матеріалу: презентація (кількість аркушів: 12)
7. Перелік публікацій:
   - наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 14-16 листопада 2018р.);
   - V Міжнародна науково-технічна Internet-конференція «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами», яка відбулась 22 листопада 2018 р. у Національному університеті харчових технологій.
8. Дата видачі завдання 5 вересня 2017 .

Календарний план

№	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою роботи	10.10.2017	
2.	Розроблення та узгодження завдання магістерської дисертації	21.10.2017	
3.	Аналіз існуючих підходів розподілу даних	10.01.2018	
4.	Тестування існуючих підходів розподілу даних	25.02.2018	
5.	Підготовка матеріалів 1-го розділу магістерської дисертації	16.04.2018	
6.	Розробка власного підходу	5.06.2018	
7.	Підготовка матеріалів 2-го розділу магістерської дисертації	27.07.2018	
8.	Реалізація запропонованого підходу	13.09.2018	
9.	Підготовка матеріалів 3-го розділу магістерської дисертації	22.10.2018	
10.	Підготовка матеріалів 4-го розділу дипломного проекту	12.10.2018	
11.	Оформлення документації дипломного проекту	23.11.2018	
12.	Підготовка графічної частини дипломного проекту	4.12.2018	
	Попередній розгляд магістерської дисертації на кафедрі	26.11.2018	

Студент _____ Даценко С.О.
(підпис) (прізвище, ініціали)

Науковий керівник дисертації _____ Петрашенко А.В.
(підпис) (прізвище, ініціали)

РЕФЕРАТ

Актуальність теми.

На сьогоднішній день MongoDB є СУБД, що часто використовується при розробці систем, орієнтованих на обробку слабко структурованих даних. Однією з причин цього є можливість використання горизонтального масштабування (шардингу) та налаштування розподілу даних, що значною мірою впливає на продуктивність роботи системи в цілому.

Тому були розроблені різні стратегії розподілу даних в кластері, такі як: стратегія палаючого будинку, стратегія гарячих точок тощо. Дані стратегії мають свої сфери застосування, свої переваги та недоліки, але загальним недоліком даних стратегій є нерівномірний розподіл даних при наявності так званих «джамбо-чанків», що в свою чергу впливає на продуктивність роботи системи.

Об'єктом дослідження є розподіл даних між серверами в кластері MongoDB.

Предметом дослідження є програмні засоби для підвищення продуктивності роботи СУБД MongoDB, які призначені для налаштування розподілу даних відповідно до допустимих відсоткових навантажень вузлів кластера, враховуючи вплив «джамбо-чанків».

Мета роботи: підвищення продуктивності роботи СУБД MongoDB.

Наукова новизна:

1. Запропоновано підхід розподілу даних, який включає в себе ідеї існуючих підходів та, на відміну від існуючих підходів, при розподілі даних враховує можливість наявності «джамбо-чанків».

2. Виконано порівняльний аналіз запропонованого підходу з існуючими, визначено в яких саме ситуаціях потрібно використовувати даний підхід, його переваги та недоліки порівняно з існуючими підходами розподілу даних.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований підхід надає можливість налаштування розподілу даних відповідно до допустимих відсоткових навантажень вузлів кластера, враховуючи вплив «джамбо-чанків». Крім того розроблені в роботі програмні засоби можуть бути використані для реалізації автоматизованого розподілу даних з заданими налаштуваннями.

Апробація роботи. Запропонований підхід був представлений та обговорений на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 14 - 16 листопада 2018 р.) та на V Міжнародній науково-технічній Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами», яка проводилась 22 листопада 2018 р. у Національному університеті харчових технологій.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів.

У першому розділі розглянуто існуючі підходи розподілу даних, їхні особливості, недоліки та переваги, розглянуто різні реалізації.

У другому розділі запропоновано підхід розподілу даних який буде вирішувати виявлену проблему.

У третьому розділі наведені алгоритмічні особливості реалізації розроблених програмних засобів для реалізації запропонованого підходу.

У четвертому розділі представлено результати тестування запропонованого підходу.

У висновках представлені результати проведеної роботи.

Магістерська дисертація представлена на 80 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: MongoDB, шардинг, ключ шардингу, шард, підхід розподілу даних, джамбо-чанк, чанк.

РЕФЕРАТ

Актуальность темы.

На сегодняшний день MongoDB является СУБД, которая часто используется при разработке систем, ориентированных на обработку слабо структурированных данных. Одной из причин этого является возможность использования горизонтального масштабирования (шардингу) и настройки распределения данных, что в значительной мере влияет на производительность работы системы в целом.

Поэтому были разработаны различные стратегии распределения данных в кластере, такие как: стратегия горящего дома, стратегия горячих точек и тому подобное. Данные стратегии имеют свои области применения, свои преимущества и недостатки, но общим недостатком данных стратегий является неравномерное распределение данных при наличии так называемых «джамбо-чанков», что в свою очередь влияет на производительность работы системы.

Объектом исследования является распределение данных между серверами в кластере MongoDB.

Предметом исследования являются программные средства для повышения производительности работы СУБД MongoDB, которые предназначены для настройки распределения данных в соответствии с допустимых процентных нагрузок узлов кластера, учитывая влияние «джамбо-чанков».

Цель работы: повышение производительности работы СУБД MongoDB.

Научная новизна:

1. Предложен подход распределения данных, который включает в себя идеи существующих подходов и, в отличие от существующих подходов, при распределении данных учитывает возможность наличия «джамбо-чанков».

2. Выполнен сравнительный анализ предложенного подхода с существующими, определено в каких именно ситуациях нужно использовать данный подход, его преимущества и недостатки по сравнению с существующими подходами распределения данных.

Практическая ценность полученных в работе результатов заключается в том, что предложенный подход предоставляет возможность настройки распределения данных в соответствии с допустимыми процентных нагрузками узлов кластера, учитывая влияние «джамбо-чанков». Кроме того разработанные в работе программные средства могут быть использованы для реализации автоматизированного распределения данных с заданными настройками.

Апробация работы. Предложенный подход был представлен и обсужден на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2018 (Киев, 14 - 16 ноября 2018) и на V Международной научно-технической Internet-конференции «Современные методы, информационное, программное и техническое обеспечение систем управления организационно-техническими и технологическими комплексами», которая проводилась 22 ноября 2018 в Национальном университете пищевых технологий.

Структура и объем работы. Магистерская диссертация состоит из введения, четырех глав и выводов.

Во введении представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов.

В первой главе рассмотрены существующие подходы распределения данных, их особенности, недостатки и преимущества, рассмотрены различные реализации.

Во втором разделе предложено подход распределения данных который будет решать обнаруженную проблему.

В третьем разделе приведены алгоритмические особенности реализации разработанных программных средств для реализации предложенного подхода.

В четвертом разделе представлены результаты тестирования предложенного подхода.

В выводах представлены результаты проведенной работы.

Магистерская диссертация представлена на 80 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: MongoDB, шардинг, ключ шардингу, шард, подход распределения данных, джамбо-чанк, чанк.

ABSTRACT

Actuality of theme.

To date, MongoDB is a database, often used in the development of systems focused on the processing of poorly structured data. One reason for this is the ability to use horizontal scaling (shading) and data distribution settings, which greatly affects the performance of the system as a whole.

Therefore, various strategies for data splitting in the cluster were developed, such as: a burning house strategy, hot spots strategy, and so on. These strategies have their own applications, their advantages and disadvantages, but the general disadvantage of these strategies is the uneven distribution of data in the presence of so-called "jumbo chunk", which in turn affects the performance of the system.

The object of the study is the distribution of data between servers in the MongoDB cluster.

The subject of the study is software tools to improve the performance of the MongoDB database, which are designed to configure the distribution of data in accordance with the permissible percentage loads of nodes of the cluster, taking into account the influence of "jumbo chunks".

The purpose of the work: to increase the productivity of the DBMS MongoDB.

Scientific novelty:

1. A data-sharing approach is proposed that incorporates the ideas of existing approaches and, unlike existing approaches, takes into account the possibility of having "jumbo chunks" when distributing data.

2. A comparative analysis of the proposed approach with existing ones is made, it is determined in what situations it is necessary to use this approach of approach, its advantages and disadvantages in comparison with existing approaches of data distribution.

The practical value of the results obtained in the work is that the proposed approach provides the possibility of adjusting the distribution of data in accordance with the permissible percentages of nodes of the cluster, taking into account the influence of "jumbo chunks". In addition, the software developed in the work can be used to implement automated data distribution with the specified settings.

Test work. The proposed approach was presented and discussed at the scientific conference of undergraduates and postgraduates "Applied Mathematics and Computing", PMK-2018 (Kyiv, November 14-16, 2018) and at the V International Scientific and Technical Internet Conference "Modern Methods, Information, software and technical support of control systems for organizational, technical and technological complexes", held on November 22, 2018 at the National University of Food Technologies.

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The *introduction* gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained

The *first chapter* examines the existing approaches to data distribution, their features, disadvantages and advantages, and discusses different implementations.

The *second chapter* proposes a data-sharing approach that addresses the problem identified.

The *third chapter,* algorithmic features of implementation of the developed software tools for implementation of the proposed approach are presented.

The *fourth chapter* presents the results of testing the proposed approach.

The *conclusions* are the results of the work.

The master's dissertation is presented on 80 sheets, contains a link to the list of used literary sources.

Keywords: MongoDB, shading, shard key, shard, data sharing approach, jumbo chunk, chunk.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Балансувальник (balancer) – внутрішній процес MongoDB, який працює в контексті кластеру та відповідає за переміщення чанків.

БД – база даних

Джамбо-прапорець (jumbo flag) – коли система розпізнає чанк як джамбо-чанк, вона в інформацію таких чанків додає поле «jumbo» та записує в нього значення «true». Це робиться для того, щоб під час подальшої роботи системи цей чанк балансувальник не пробував переміщувати та до нього не застосовувались операції поділу, адже вони будуть неуспішними.

Джамбо-чанк (jumbo chank) – це чанк, розмір якого перевищує максимально допустимий та його не можна поділити, оскільки в ньому містяться дані лише за одним значенням ключа.

Зона (zone) – це групування документів базуючись на діапазоні значень ключа розподіленої колекції.

Ключ шардингу (shard key) - це поле за яким MongoDB розподіляє дані в кластері.

НБД – назва бази даних

НЗ – назва зони

НК – назва колекції

НПК – назва поля-ключа

НШ – назва шарда

Поле ObjectId (поле «_id») – спеціальне 12-байтний BSON тип який гарантує унікальність документів в колекції.

СУБД – система керування базами даних

Хешовий ключ шардингу (Hashed Shard Key) – особливий вид ключа шардингу, який використовує хешове значення ключа шардингу для розподілу даних між членами кластеру.

Цілenаправлені операції (Targeted Operations) – операції в яких вже відомо де знаходяться дані (тобто відбувається звернення лише до серверів з необхідними даними).

Цілочисельний тип даних – це тип даних, значеннями якого є лише цілі числа.

Чанк (chunk) – блок даних в якому зберігаються дані певного діапазону значень ключа шардингу.

Шард (shard) – це один mongod-сервер або набір mongod-серверів об’єднаних у набір реплік, який зберігає частину даних кластеру.

Шардинг (Sharding) – це архітектура бази даних, яка розподіляє дані за значеннями ключів шарднигу між двома або більше шардами в базі даних.

BigData — набори інформації (як структурованої, так і неструктурованої) настільки великих розмірів, що традиційні способи та підходи не можуть бути застосовані до них.

Broadcast Operations – операції під час яких відбувається звернення до всіх існуючих шардів у системі.

BSON — комп'ютерний формат обміну даними. Це двійкова форма представлення простих структур даних і асоціативних масивів (які називають об'єктами або документами).

ВСТУП

Кожна сучасна система має власну базу даних (БД), яка управляється різними СУБД. Загалом СУБД поділяють на два типи: реляційні (наприклад: Oracle Database, Microsoft SQL Server, MySQL, IBM DB2, IBM Informix, SAP Sybase Adaptive Server Enterprise, SAP Sybase IQ, Teradata) та NoSQL, які в свою чергу залежно від їх масштабування, систем збереження даних, моделей даних та запитів розділяють на такі під-типи:

- ключ/значення – це база даних, яка використовує ключ для доступу до значення, яке зберігається в ньому, прикладами таких СУБД є: BerkleyDB, MemcacheDB, Redis, Riak, Amazon DynamoDB тощо [1];
- документо-орієнтовані – це бази даних призначені для зберігання ієрархічних структур даних, прикладами таких СУБД є: CouchDB, Couchbase, MarkLogic, MongoDB, eXist, BerkleyDB XML тощо [1];
- стовпчиково-орієнтовані – це бази даних в яких дані зберігаються у вигляді розрідженої матриці, рядки та стовпчики якої використовуються в якості ключів, прикладами таких СУБД є: Apache HBase, Apache Cassandra, Apache Accumulo, Hypertable, SimpleDB тощо [1];
- графові – це бази даних реалізація мережевої моделі яких має вигляд графу та його узагальнень, прикладами таких СУБД є: Neo4j, OrientDB, AllergoGraph, InfiniteGraph, FlockDB, Titan тощо [1].

З наведеного вище опису різних СУБД можна помітити, що MongoDB відноситься до NoSQL БД документно-орієнтованого типу, які часто використовуються у системах управління вмістом, видавничій справі, документальному пошуку, інформаційно-моніторингових системах тощо. Основною причиною цього є її висока гнучкість, багатий функціонал та висока швидкодія, порівняно з іншими СУБД, які неорієнтовані лише на швидкодію [2].

Однією з переваг NoSQL СУБД над реляційними СУБД є можливість горизонтального масштабування та налаштування розподілу даних, що значною мірою впливатиме на продуктивність роботи системи в цілому.

В MongoDB є три базові стратегії розподілу даних між серверами:

- розподіл за звичайним ключем;
- розподіл за хешовим ключем;
- розподіл по діапазонах.

Крім базових стратегій існують й інші (такі як: стратегія палаючого будинку, стратегія гарячих точок тощо), але потрібно відзначити, що дані методи (стратегії) базуються на взаємодії однієї або декількох базових стратегій та додаткових процедур, якщо вони потрібні.

Всі існуючі стратегії мають свої сфери застосування, свої переваги та недоліки, але підчас їх тестування було виявлено, що при наявності так званих «джамбо-чанків» розподіл даних стає нерівномірним, якщо порівнювати розмір даних на кожному сервері, а це в свою чергу впливатиме на продуктивність роботи системи.

Під час пошуку можливих рішень даної проблеми було знайдено:

- алгоритм переміщення «джамбо-чанків» з одного серверу на інший;
- поради по зміні ключів так, щоб ймовірність появи «джамбо-чанків» була меншою – хоча це не призведе до повного зникнення «джамбо-чанків».

Таким чином метою даної роботи стало створення нової стратегії розподілу даних, яка буде враховувати можливість наявності «джамбо-чанків», та розробити програмні засоби для її реалізації.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ РОБОТИ СУБД MONGODB

Кожна сучасна система, яка працює з BigData, має свою власну БД та підсистему управління нею, тобто СУБД. Дана підсистема відповідає за збереження та зчитування даних, але потрібно підкреслити, що для роботи сучасних систем ресурсів, які може надати один комп’ютер недостатньо, тому була створена техніка горизонтального-масштабування, яка надає можливість поєднувати комп’ютери в одну єдину систему.

Якщо в випадку використання однієї машини для роботи системи на її продуктивність буде впливати те як саме вона спроектована, то в випадку використання декількох машин – на її продуктивність будуть впливати: архітектура системи, як розподілені дані між машинами та швидкість зчитування й запису даних. Тому на сьогоднішній день при розробці таких систем особливу увагу приділяють саме розподілу даних.

MongoDB є системою управління базами даних, яка реалізує можливість використання техніки горизонтального-масштабування (шардингу), та для розподілу даних в даній системі були створені різні методи (підходи), які мають свої переваги та недоліки.

1.1. Існуючі підходи розподілу даних у MongoDB

Розглядаючи існуючі підходи розподілу даних у MongoDB, їх можна умовно поділити на дві категорії: базові та складені. До базових підходів належать такі методи розподілу:

1) розподіл за звичайним ключем (Ranged Sharding) [3, 4];
2) розподіл за хешовим ключем (Hashed Sharding)[4, 5];
3) розподіл з використанням зон [4, 6].

Найбільш відомими складеними методами розподілу є:

- стратегія палаючого будинку [4];
- стратегія гарячих точок [4].

1.1.1. Розподіл за звичайним ключем (Ranged Sharding)

Даний підхід полягає в тому, що перед розподілом вибирається поле – яке буде виступати в якості ключа, після чого влаштований розподільник (балансувальник) розподілить дані між шардами, таким чином, щоб кількість чанків на кожному шарді була майже однакова. Потрібно зауважити, що розподіл буде відбуватись за дійсними значеннями поля-ключа, саме ж поле може бути числового (цілочисельного або з плаваючою комою) або строкового (рядкового (string) або символьного (char)) типу [3, 4].

Розподіл за звичайним ключем передбачає поділ даних на сусідні діапазони, визначені значеннями ключа шардингу. У цій моделі, документи з «близькими» значеннями ключа, ймовірно, будуть знаходитись в одному чанку або на одному шарді – це надає можливість більш ефективно опрацьовувати запити на діапазонний пошук (зчитування), за рахунок використання цілеспрямованого пошуку. Проте ефективність зчитування та запису може зменшитись при поганому виборі ключа шардингу [3, 4].

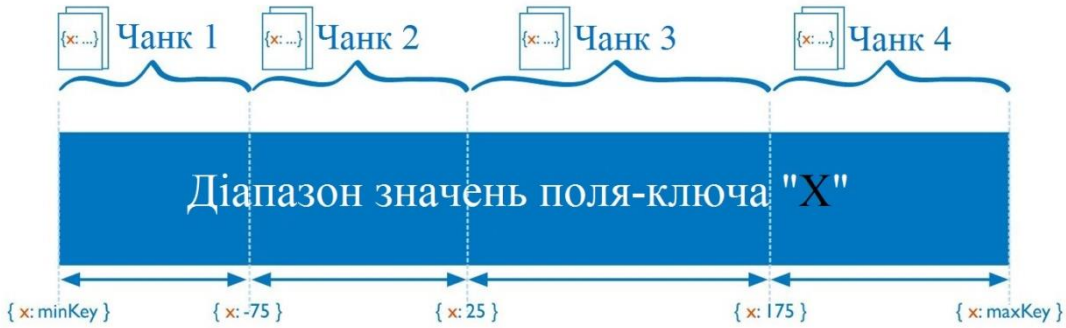

Рисунок 1.1 – Приклад розподілу за звичайним ключем

На рис. 1.1 зображено як відбувається шардинг за звичайним ключем. У даному випадку:

1) в якості ключа шардингу виступає поле «X», воно є цілочисельним;
2) дані було поділено на 4 сусідні діапазони ([minKey (-∞); -75), [-75; 25), [25; 175) та [175; maxKey (+∞))), усі ці дані було записано в

відповідні чанки. Потрібно зазначити, що після поділу балансувальник розподілить дані чанки між усіма шардами;

3) якщо буде запит на пошук даних за діапазоном, наприклад, [30; 57], то система буде лише працювати з 3 чанком (з шардом на якому він знаходиться), на відміну від пошуку за цим же діапазон при розподілі за хешовим ключем (див пункт 1.1.2.).

Даний підхід буде найбільш ефективно працювати коли значення ключа шардингу відповідають таким критеріям [3, 4]:

- велика різноманітність;
- низька повторюваність;
- немонотонно змінюється.

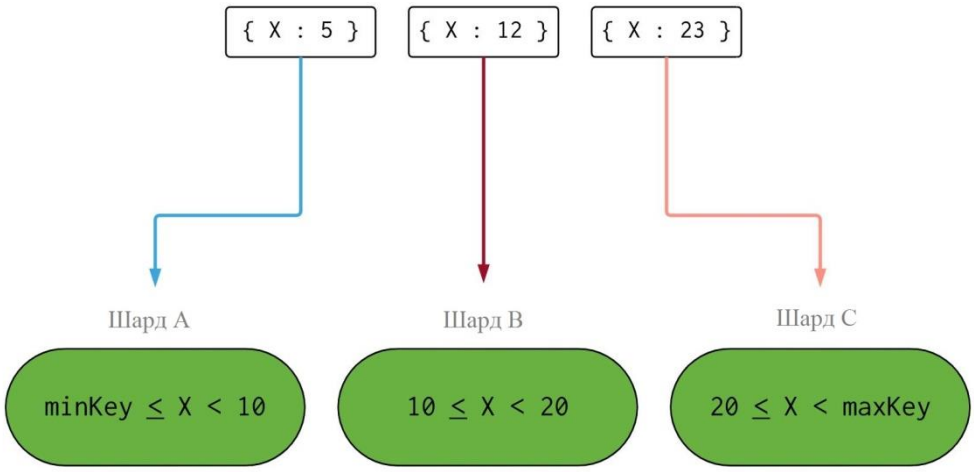

Рисунок 1.2 – Ілюстрація розподілу даних в кластері при використанні підходу Ranged Sharding

На рис. 1.2 проілюстровано розподіл даних в кластері при використанні поля «X» в якості ключа шардингу. Якщо значення поля «X» відповідають всім критеріям, то розподіл даних буде відбуватись за даною схемою.

Для реалізації даного підходу потрібно:

1) надати дозвіл на розподіл заданої бази даних командою:

> sh.enableSharding("НБД") [7];

2) якщо в колекції уже існують дані, то потрібно створити індекс для поля-ключа командою:

> *db.НК.createIndex({"НПК" : 1})* [7];

3) надати дозвіл на розподіл даних заданої колекції командою:

> *sh.shardCollection("НБД.НК»", { ({"НПК" : 1})* [7];

Таблиця 1.1 - Результати тестування розподілу за звичайним ключем

	1 000 000 записів, 100% покриття			2 000 000 записів, 10% покриття			
	Шард	К-сть чанків	Розмір, Мб	Шард	К-сть чанків	Розмір, Мб	Jumbo chunks
Колекція: thermal	a	31	17,4786	a	52	43.3344	22
	b	31	19,7993	b	52	45.4037	22
	c	31	20,5021	c	52	48.6021	21
	d	31	16,3618	d	53	56.0095	35
Колекція: gps	a	34	20,5907	a	129	81.2364	0
	b	34	20.6664	b	129	95.3743	0
	c	35	22.1621	c	130	94.8204	0
	d	34	20.5041	d	130	62.1154	0

В таблиці 1.1 наведені результати тестування розподілу за звичайним ключем. Тестування проводилось в два етапи: спочатку було записано 1 000 000 документів до колекції з покриттям значень ключів – 100%, після – 2 000 000 записів з покриттям – 10%, для цього використовувався алгоритм додання документів до колекцій (див підрозділ 3.12). Також максимально допустимий розмір чанку був змінений з розміру по замовчуванню (64Мб) на 1Мб. Тестування відбувалось на 2х колекціях «thermal» та «gps». Після завершення кожного етапу тестування використовувався алгоритм поділу чанків (див. підрозділ 3.5), а розміри даних та кількість «джамбо-чанків» отримувались з результатів роботи алгоритму перевірки використаної пам’яті.

У колекції «thermal» поле-ключ є цілочисельним та має діапазон значень від 0 до 1000, кожний запис в собі зберігає:

- поле пристрою (цілочисельне);
- поле температури (число з плаваючою комою);
- поле дати (типу Date/строкове).

У колекції «gps» поле-ключ є також цілочисельним, але має діапазон значень від 0 до 5000, кожний запис в собі зберігає:

- поле пристрою (цілочисельне);
- поле координат (масив з 2х елементів цілочисельного типу);
- поле дати (типу Date/строкове).

З результатів видно, що:

1) балансувальник розподіляє чанки між шардами рівномірно;
2) навіть при наявності однакої кількості розмір пам’яті використаний на їх збереження є різним. Явно це видно в результатах колекції «gps» після додання 2 000 000 записів на шарді «c» та «d» по 130 чанків, але реальний розмір даних на кожному шарді відрізняється в 33 Мб;
3) з другого пункту випливає, що навіть при ситуаціях, коли на шардах майже однакова кількість чанків – це не означатиме, що розподіл є рівномірним, іншим доказом цього є те, що в результаті даних тестів для колекції «thermal» на кожному шарді використаний об’єм пам’яті мав становити приблизно 48,3 Мб (відповідає цьому лише один шард з чотирьох), а для колекції «gps» - 83,4 Мб (немає відповідностей);
4) в колекції «thermal» в результаті на шарді «d» знаходиться 53 чанки розмір яких становить 56 Мб, хоча максимально допустимим розміром для такої кількості має бути 53 Мб – а це різниця в 3 Мб. Це пов’язано з тим, що на шарді наявні «джамбо-чанки», у даному випадку розмір кожного чанку складав лише приблизно 1,3 Мб, але при подальшій роботі системи їх розміри будуть збільшуватись. Тому в подальшому навіть при рівномірному розподілені балансувальником чанків між шардами самі дані рівномірно розподілені не будуть.

Переваги розподілу за звичайним ключем:

- проста реалізація;

- підвищена ефективність пошуку за діапазонами значень (при відповідності значень поля-ключа вказаним критеріям).

Недоліки даного підходу були описані в результатах тестування вище, а потрібно добавити ще один недолік – це неможливість робити розподіл даних відсотковим/пропорційним (наприклад, щоб на 1й шард надходили 25% всіх даних, на другий – 35%, на третій та четвертий – по 20%), адже в системі можуть бути як більш потужні машини так і менш потужні.

1.1.2. Розподіл за хешовим ключем (Hashed Sharding)

Даний підхід схожий з попереднім, але відрізняється тим, що замість розподілу за реальними значеннями ключа, розподіл буде відбуватись за їх хешовими значеннями/індексами – за генерування яких відповідає «Хешова функція (Hash Function)» [4, 5].

Розподіл за хешовим ключем забезпечує більш рівномірний розподіл даних, але ціною цього є зменшення ефективності діапазонного пошуку, адже на відміну від розподілу за звичайним ключем, в даному розподілі документи з «близькими» значеннями ключа ймовірніше всього будуть знаходитись в різних чанках, тому при діапазонному пошуку система буде використовувати Broadcast пошук. [4, 5]

Рисунок 1.3 – Розподіл даних між чанками при використанні Hashed Sharding

На рис. 1.3 проілюстровано як відбувається розподіл даних при використанні Hashed Sharding: при надходженні документи проходять обробку «Хешовою функцію» де відбувається обчислення їх хешових значень після чого вони додаються до певного чанку, та продемонстровано, що документами з «сусідніми» значеннями поля-ключа можуть знаходитись в різних чанках [4, 5].

Також розробники MongoDB не рекомендують при використанні даного підходу полем-ключем вибирати поля тип яких є числа з плаваючою комою. Причиною цього є те, що перед виконанням «Hash Function» дані значення будуть приводитись до цілочисельного типу, тому, наприклад, значення 2.3, 2.2 та 2.9 матимуть однаковий хешовий індекс. Щоб побачити яке хешове значення відповідатиме реальному значенню ключа можна скористатись командою *convertShardKeyToHashed()* [7], який присутній лише в версіях MongoDB 4.0 та вище [4, 5].

Поле яке вибирається в якості ключа має мати гарну потужність [8] або велику кількість різних значень. Хешові ключі будуть ідеально підходити для полів, які змінюються монотонно (наприклад, поле «ObjectId») [4, 5].

Для реалізації даного підходу потрібно:

1) надати дозвіл на розподіл даних заданої бази даних командою:

> *sh.enableSharding("НБД")* [7];

2) якщо в колекції уже існують дані, то потрібно створити індекс для поля-ключа командою:

> *db.НК.createIndex({"НПК" : "hashed"})* [7];

3) надати дозвіл на розподіл даних заданої колекції командою:

> *sh.shardCollection("НБД.НК", { ({"НПК" : "hashed"})* [7];

Для розподілу за хешовим ключем також були проведені аналогічні тести, результати наведені в таблиці 1.2.

Таблиця 1.2 - Результати тестування розподілу за хешовим ключем

	1 000 000 записів, 100% покриття			2 000 000 записів, 10% покриття		
	Шард	К-сть чанків	Розмір, Мб	Шард	К-сть чанків	Розмір, Мб
Колекція: thermal	a	33	16,7246	a	71	41,3166
	b	33	14,8266	b	71	46,7669
	c	33	16,4644	c	71	51,2311
	d	33	16,5836	d	71	44,3934
Колекція: gps	a	19	26,1501	a	72	62,5836
	b	19	19,1774	b	72	65,175
	c	17	23,4444	c	72	61,2036
	d	19	24,0318	d	71	68,3823

У таблиці 1.2 наведені результати тестування розподілу за хешовим ключем. Тестування проводилось в два етапи: спочатку було записано 1 000 000 документів до колекції з покриттям значень ключів – 100%, після – 2 000 000 записів з покриттям – 10%, для цього використовувався алгоритм додання документів до колекцій (див підрозділ 3.12). Також максимально допустимий розмір чанку був змінений з розміру по замовчуванню (64Мб) на 1Мб. Тестування відбувалось на 2х колекціях «thermal» та «gps». На відміну від тестування розподілу за звичайним ключем, у тестуванні розподілу за хешовим ключем, після завершення кожного етапу тестування не використовувався алгоритм поділу чанків (див. підрозділ 3.5) через те, що він не працює з хешовим ключем, аналогічно алгоритм перевірки використаної пам’яті. (див. підрозділ 3.13). Розмір даних на кожному шарді отримувався з результатів наступної команди:

> *db.НК.stats()* [7]

Результатом даної команди є інформація про стан колекції, але в неї не включено інформації про «джамбо-чанки». Також у результаті вказаний середній розмір документів колекції (він є однаковим для всіх тестувань),

для колекції «thermal» він становить 64 Байти, а для колекції «gps» - 88 Байтів.

Аналогічно тестуванню Ranged Sharding, у колекції «thermal» поле-ключ є цілочисельним та має діапазон значень від 0 до 1000, кожний запис в собі зберігає:

- поле пристрою (цілочисельне);
- поле температури (число з плаваючою комою);
- поле дати (типу Date/строкове).

У колекції «gps» поле-ключ є цілочисельним, але має діапазон значень від 0 до 5000, кожний запис в собі зберігає:

- поле пристрою (цілочисельне);
- поле координат (масив з 2х елементів цілочисельного типу);
- поле дати (типу Date/строкове).

З результатів видно, що на першому етапі тестування:

1) у колекції «thermal» поріг поділу чанків був досягнений та виконався автоматичний поділ (у колекції «gps» поріг поділу був також досягнений, але автоматичний поділ не відбувся – причина цього описана в підрозділі 2.2);
2) розподіл за кількістю чанків не гарантує рівномірний розподіл даних:
   - на шардах «a» та «b» однакова кількість чанків, але розмір відрізняється – у колекції «gps» на 7 Мб, а в колекції «thermal» на 2 Мб.;
   - розмір даних на кожному шарді колекції «thermal» мав бути приблизно 16,15 Мб, а в колекції «gps» - 23,2 Мб (є невідповідності).

З результатів видно, що на другому етапі тестування:

1) у колекції «thermal» поріг поділу чанків був досягнений та виконався автоматичний поділ (у колекції «gps» поріг поділу був

також досягнений, але автоматичний поділ не відбувся – причина цього описана в підрозділі 2.2);

2) розподіл за кількістю чанків не гарантує рівномірний розподіл даних:
   - у колекціях «thermal» та «gps», розмір даних на деяких шардах відрізняються (різниця в розмірі перевищує 1 Мб, тобто на таких шардах даних знаходиться більше щонайменше на один чанк), хоча сам розмір знаходиться в допустимих рамках (відповідно кількості чанків);
   - розмір даних на кожному шарді колекції «thermal» мав бути приблизно 46 Мб, а в колекції «gps» - 64,34 Мб (є невідповідності).

Потрібно зазначити, що навіть при розподілі за хешовим ключем «джамбо-чанки» також можуть з’являтись, докази цього зображений на рис. 1.4 та рис. 1.5.

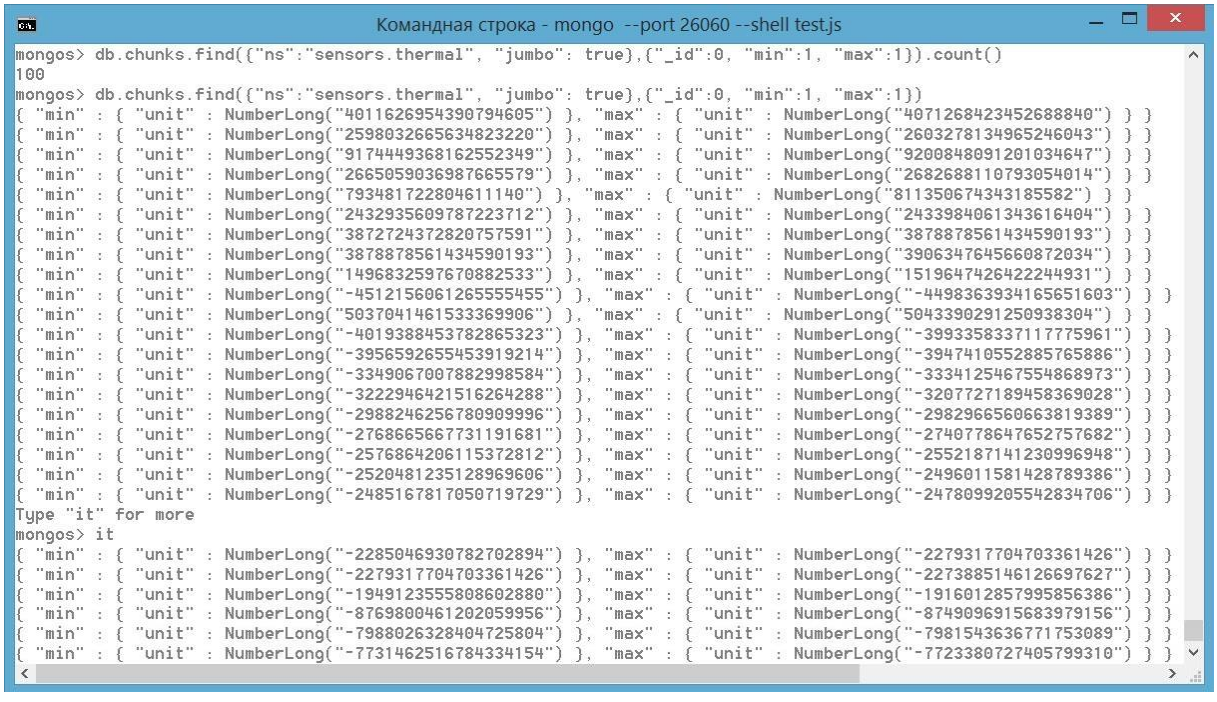

Рисунок 1.4 – Підтвердження можливої наявності «джамбо-чанків» при розподілі за хешовим ключем. У колекції «chunks» при пошуку існуючих «джамбо-прапорців» для колекції «thermal»

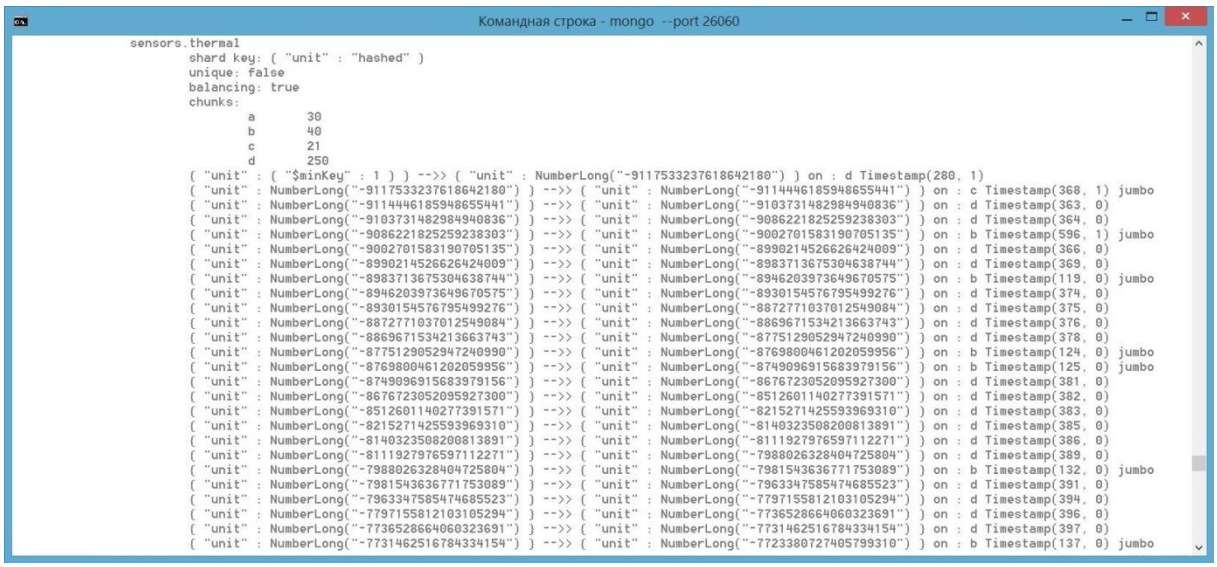

Рисунок 1.5 – Підтвердження можливої наявності «джамбо-чанків» при розподілі за хешовим ключем, через статус шардингу

Переваги розподілу за хешовим ключем:

- проста реалізація;
- більш рівномірний розподіл порівняно з розподілом за звичайним ключем.

Недоліки даного підходу були описані в результатах тестування вище та в описі підходу, але потрібно добавити ще такі недоліки:

– неможливість робити розподіл даних відсотковим/пропорційним (наприклад, щоб на 1й шард надходили 25% всіх даних, на другий – 35%, на третій та четвертий – по 20%), адже в системі можуть бути як більш потужні машини так і менш потужні;

– неможливість наглядно дізнатись місце знаходження даних (можливо дізнатись місце положення тільки при використанні ціле-направленого пошуку з використанням команди «explain» [7]).

1.1.2.1. Порівняння Hashed та Ranged Sharding при монотонно зростаючому ключі

У колекції, яка для шардингу використовує монотонно зростаюче поле-ключ «X», при використанні Ranged Sharding розподіл даних буде

відбуватись відповідно схеми зображеної на рис. 1.6, дані будуть записуватись в чанк з maxKey [4, 5].

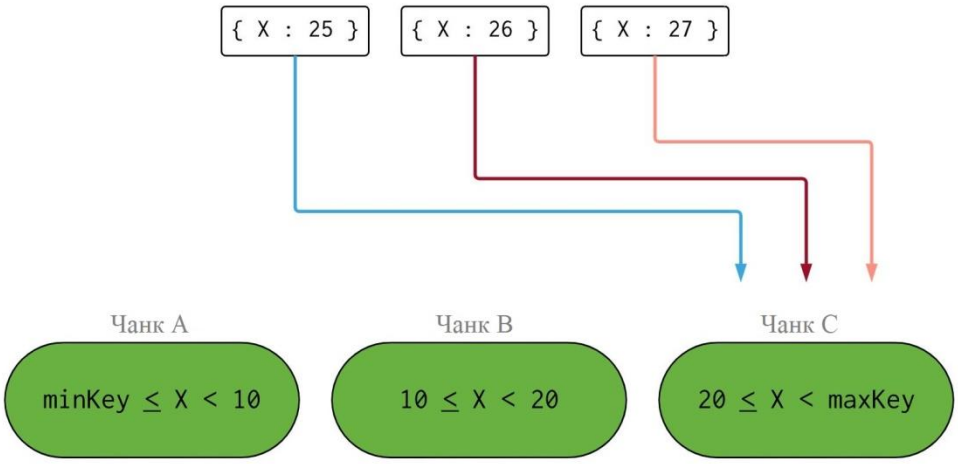

Рисунок 1.6 – Схема розподілу документів при використанні Ranged Sharding

Такий розподіл обмежує запис лише до одного шарду в один чанк, що зменшує або усуває перевагу розподілених записів у кластері. Для уникнення даної проблеми можна використовувати Hashed Sharding (рис. 1.7) [4, 5].

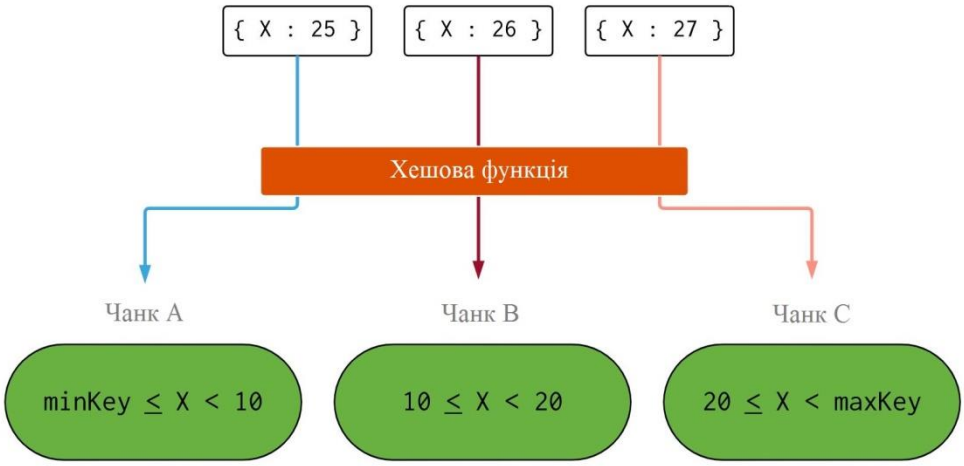

Рисунок 1.7 - Схема розподілу документів при використанні Hashed Sharding

У такому випадку дані будуть розподілятись більш рівномірно по всьому кластері та ефективність запису документів буде вища [4, 5].

1.1.3. Розподіл з використанням зон

Інколи під час налаштування розподілу виникає потреба призначити шарду або групі шардів певний діапазон значень ключа, саме для цього в MongoDB використовують зони. Особливостями зон [4, 6] є:

1) кожна зона має мати хоча б один підключений шард до неї;
2) один і той же шард може бути підключеним до різних зон;
3) діапазони існуючих зон не мають перетинатись, тобто неможливе існування зон зі спільним діапазоном значень (наприклад, не можливе одночасне існування зони «A» з діапазоном [10; 20) та зони «B» з діапазоном [15; 25);
4) ключ шардингу може бути як хешовим так і звичайним, особливості розподілу будуть відповідати характеристикам ключа шардингу;
5) на шарди, до яких підключені зони, неможна записувати дані, які не знаходяться в діапазонах підключених зон;
6) якщо ключем шардингу є складений ключ, то при створені зони потрібно в полі-ключа вказувати всі поля ключа.

Зони часто використовуються коли потрібно [4, 6]:

- призначити певну підмножину даних певному шарду;
- для страхування, щоб найбільш релевантні дані знаходились на найбільш близькому (географічно) сервері;
- давати більше/менше навантаження на більш/менш потужні сервери.

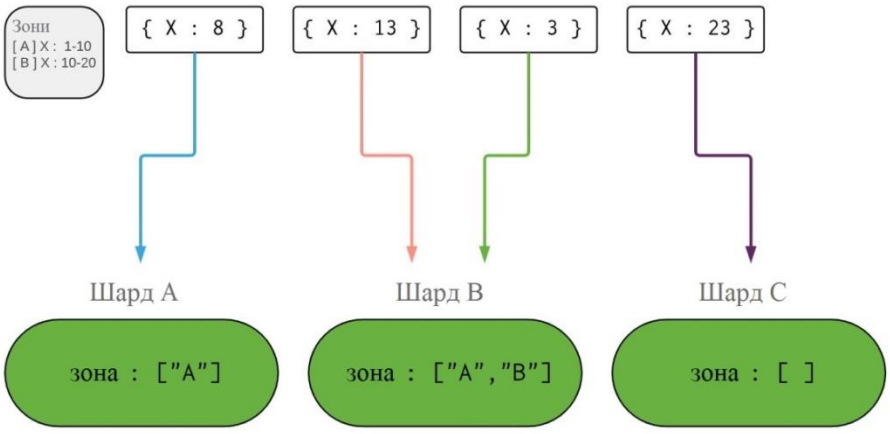

Рисунок 1.8 – Розподіл з використанням зон

На рис. 1.8 схематично продемонстровано як будуть розподілятись дані при використанні зон. У даному випадку в кластері для розподіленої колекції існує дві зони («A» та «B») яким відповідають такі діапазони: [1; 10) та [10; 20). Сам же кластер складається з трьох шардів: «A», «B» та «C». До шарду «A» підключена зона «A», до шарду «B» - зони «A» та «B», а до шарду «C» не підключено жодної зони, тому на цей шард будуть записуватись усі дані, які не входять до існуючих зон [4, 6].

Для реалізації даного підходу потрібно [4, 6]:

1) дозволити шардинг колекції, ключ може бути як звичайним (див. пункт 1.1.1) так і хешовим (див. пункт 1.1.2);
2) до вибраного шарду підключити зону командою:
   > sh.addShardToZone("НШ", "НЗ") [7];
3) призначити певний діапазон зоні командою:
   > sh.updateZoneKeyRange("НБД.НК", {"НПК": «мін. значення ключа (нижня границя зони)» }, { "НПК": «макс. значення ключа (верхня границя зони)» }, "НЗ") [7].

Для роботи з зонами також можна використовувати команди *sh.removeRangeFromZone()* [7] та *sh.removeShardFromZone()* [7] для зміни діапазону значень зони та відключення шарду від зони [4, 6].

Інформація та налаштування всіх існуючих зон зберігаються в колекції «tags» бази даних «config» [4, 6].

Таблиця 1.3 - Результати тестування розподілу з використанням зон

	1 000 000 записів, 100% покриття			2 000 000 записів, 10% покриття			
	Шард	К-сть чанків	Розмір, Мб	Шард	К-сть чанків	Розмір, Мб	Jumbo chunks
Колекція: thermal	a	35	15,269	a	122	137,3392	100
	b	34	15,2661	b	34	15,2661	0
	c	34	15,2376	c	34	15.2376	0
	d	35	15,2625	d	35	15,2625	0

Таблиця 1.3 (продовження)

	1 000 000 записів, 100% покриття			2 000 000 записів, 10% покриття			
	Шард	К-сть чанків	Розмір, Мб	Шард	К-сть чанків	Розмір, Мб	Jumbo chunks
Колекція: gps	a	47	20,9184	a	360	188,7651	0
	b	49	21,051	b	49	21,051	0
	c	49	21,0002	c	49	21,0002	0
	d	47	20,9537	d	47	20,9537	0

Тестування проходило аналогічно тестуванню Ranged Sharding, максимально допустимий розмір чанків дорівнює 1 Мб, розподіл відбувався за звичайним ключем. Після кожного етапу тестування виконувались алгоритми поділу чанків (див. підрозділ 3.5) та перевірки використаної пам’яті (див. підрозділ 3.13). Для колекції «thermal» зони були створені з такими діапазонами: [1; 251) , [251; 501) , [501; 751) та [751; 1001), а для колекції «gps» - [1; 1251) , [1251; 2501) , [2501; 3751) та [3751; 5001). У таблиці 1.3 наведено результати тестування з яких видно, що:

1) при рівномірному надходженні даних розподіл буде залишатись рівномірним як за кількість чанків так і за кількістю використаної пам’яті;
2) при нерівномірному надходженні даних про рівномірний розподіл можна не говорити, оскільки дані будуть записуватись лише в відповідні їм зони. Тому буде виникати потреба змінити границі діапазонів зон, або весь підхід розподілу;
3) вплив «джамбо-чанків» на розподіл продемонстровано в результатах колекції «thermal»: 122 чанки мають розмір в 137,3 Мб, хоча такий розмір мають мати 138 чанків – різниця в 16 чанків. Це доводить, що розподілення даних лише за кількістю чанків не завжди є рівномірним.

1.1.3.1. Робота балансувальника при використанні зон

В звичайних умовах, коли відсутні зони, балансувальник завжди намагається рівномірно розподіли чанки між усіма шардами в кластері. Але при використанні зон його принцип роботи дещо інший. Якщо при перевірці чанк [6]:

- належить певній зоні, але не знаходиться на відповідному/-их йому шарді/-ах, то він його перемістить;
- не належить жодній з зон, то він буде розміщений на шарді у якого немає підключеної зони;
- знаходиться на шарді з підключеною зоною, але не належить їй, то він буде переміщений.

1.1.3.2. Розподіл за хешовим ключем з використанням зон

При використанні зон на хешовому ключі шардингу, зони будуть покривати діапазони хешових значень, а не дійсних. Наприклад, взявши документ значення ключа шардингу в якому рівне 1 та при наявності зони нижня границя якої рівна 1, це не буде означати, що даний документ буде належати даній зоні [6].

Таким чином зони, які покривають певні діапазони значень хешового ключа, можуть мати несподівану поведінку. Але якщо потрібно лише гарантувати, що всі записи будуть знаходитись на одному шарді, то можна створити зону границі якої будуть (minKey (-∞); maxKey (+∞)) [6].

1.1.4. Стратегія палаючого будинку (The Firehose Strategy)

Як було уже написано в пункті 1.1.3, інколи з’являється потреба збільшити, або зменшити навантаження на певні/-ий шард/-и, саме для цього можна використовувати дану стратегію. Загалом дана стратегія базується на використанні двох базових підходів, а саме: розподіл за звичайним ключем та розподіл з використанням зон [4].

Краще всього дану стратегію можна описати на прикладі: припустимо в нашій системі є декілька серверів (шардів), але один з них може опрацьовувати в 10 раз більше даних (хоча навантаження на нього таке ж як і у всіх інших). У даному випадку звісно ж доцільним буде збільшити навантаження на даний сервер в 10 раз, це буде досягатись шляхом направлення всіх нових записів саме на цей шард, при цьому всі інші сервери будуть відповідати за зчитування вже існуючих даних. Для того, щоб це зробити ми маємо вибрати в якості поля-ключа поле діапазон значень якого постійно монотонно змінюється (зростає). Наприклад, таким полем може бути поле «_id», або поле дати, яке части присутнє в документах. Після чого потрібно створити зону в яку будуть входити всі «нові» записи, ця зона має мати такий діапазон значень – [«нинішнє макс. значення ключа»; maxKey (+∞)), та підключити її до потрібного сервера [4].

Але це, ще не все, оскільки, крім виконання даних дій потрібно, ще написати програмний засіб, який би при кожному зростанні діапазону значень ключа змінював нижню границю створеної зони. Адже без цього, у подальшому в кластері всі «нові» дані, отримані після створення зони, зможуть знаходитись лише на даному шарді, а потрібно, щоб вони стали «старими» даними та були розподілені між іншими шардами. Для цього потрібно використати наступні команди [4]:

```
> use config
> var zone = db.tags.findOne({"ns" : "НБД.НК", "max" : {"НПК" : MaxKey}})
> zone.min.shardKey = «нинішнє макс. значення ключа»
> db.tags.save(zone)
```

Даний програмний код виконує наступні операції:

1) підключення до бази даних «config»;
2) знаходження інформації та налаштувань про створену зону, запис всіх даних в змінну «zone»;
3) редагування значення нижньої границі зони;

4) зміна налаштувань зони.

Таким чином при появі «нового» значення ключа, нижня границя зони має змінитись, а всі існуючі дані будуть відповідно розподілені [4].

Недоліком даної стратегії є те, що у випадку, коли «нових» даних було занадто багато, розмір чанка перевищив максимально допустимий, то навіть після зміни нижньої границі зони, даний чанк не буде переміщений. Причиною цього є те, що він буде класифікований як «джамбо-чанк», а їх переміщувати без зміни максимально допустимого розміру чанків недозволено. Тому автори даної стратегії не рекомендують її використання в разі відсутності високоефективного серверу та рекомендують при не використанні зон, не використовувати постійно зростаюче поле в якості ключа шардингу [4].

1.1.5. Стратегія гарячих точок (Multi-Hotspot Strategy)

Автономні сервери найбільш ефективні коли виконують висхідні записи, але це суперечить шардингу, оскільки шардинг є найбільш ефективний коли всі записи розповсюдженні по всьому кластеру. Дана стратегія по суті створює гарячі точки по всьому кластеру – оптимально для кожного шарду – таким чином записи будуть відбуватись по всьому кластеру, але в шарді будуть відбуватись висхідні записи [4].

Для реалізації даної стратегії потрібно використовувати складений ключ перше поле якого є поле з низькою різноманітністю значень, а друге – постійно зростаюче. Це буде виглядати так (рис 1.9): у кожного чанка кількість значень першого поля буде обмеженою, а в другому полі спочатку буде діапазон (minKey (-∞) ; maxKey (+∞)), але при подальшій роботі системи відбудеться його розбиття на менші діапазони [4].

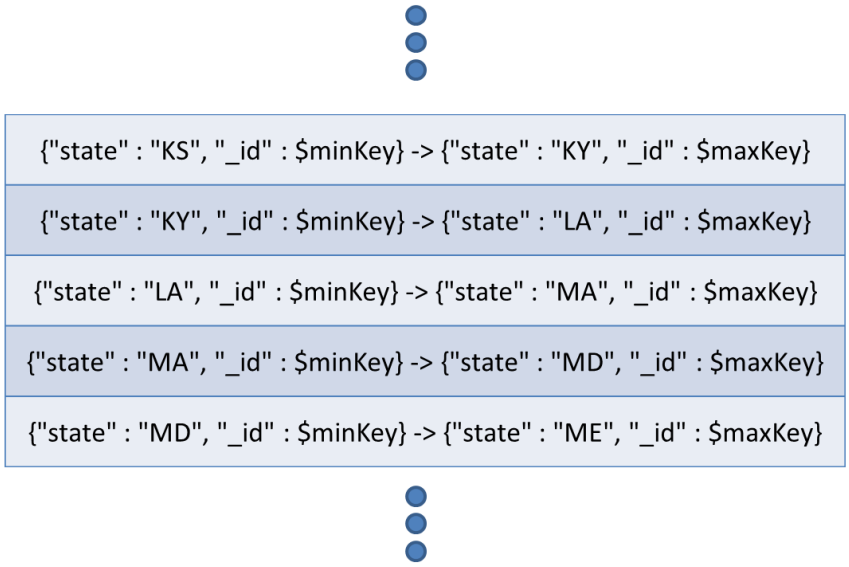

Рисунок 1.9 – Підмножина чанків. Кожен чанк має два поля відповідно до стратегії гарячих точок


```
{"state" : "MA", "_id" : ObjectId("511bf9e17d55c62b2371f1d")}
{"state" : "NY", "_id" : ObjectId("511bf9e17d55c62b2371f1e")}
{"state" : "CA", "_id" : ObjectId("511bf9e17d55c62b2371f1f")}
{"state" : "NY", "_id" : ObjectId("511bf9e17d55c62b2371f20")}
{"state" : "MA", "_id" : ObjectId("511bf9e17d55c62b2371f21")}
{"state" : "MA", "_id" : ObjectId("511bf9e17d55c62b2371f22")}
{"state" : "NY", "_id" : ObjectId("511bf9e17d55c62b2371f23")}
{"state" : "CA", "_id" : ObjectId("511bf9e17d55c62b2371f24")}
{"state" : "CA", "_id" : ObjectId("511bf9e17d55c62b2371f25")}
```


Рисунок 1.10 – Приклад доданих документів

Chunk:	{"state" : “CA", "_id" : $minKey} -> {"state" : “CO", "_id" : $maxKey}
{"state" : "CA", "_id" : ObjectId("511bf9e17d55c62b2371f1f")}	
{"state" : "CA", "_id" : ObjectId("511bf9e17d55c62b2371f24")}	
{"state" : "CA", "_id" : ObjectId("511bf9e17d55c62b2371f25")}	

Chunk:	{"state" : “MA", "_id" : $minKey} -> {"state" : “ME", "_id" : $maxKey}
{"state" : "MA", "_id" : ObjectId("511bf9e17d55c62b2371f1d")}	
{"state" : "MA", "_id" : ObjectId("511bf9e17d55c62b2371f21")}	
{"state" : "MA", "_id" : ObjectId("511bf9e17d55c62b2371f22")}	

Chunk:	{"state" : “NY", "_id" : $minKey} -> {"state" : “OH", "_id" : $maxKey}
{"state" : "NY", "_id" : ObjectId("511bf9e17d55c62b2371f1e")}	
{"state" : "NY", "_id" : ObjectId("511bf9e17d55c62b2371f20")}	
{"state" : "NY", "_id" : ObjectId("511bf9e17d55c62b2371f23")}	

Рисунок. 1.11 – Додані документи поділені по чанках

Другою частиною ключа є зростаюче поле. Це означає, що в чанку, значення завжди зростають (приклад наведено на рис. 1.10). Таким чином, якщо мати по одному чанку на шард, то отримуємо ідеальні налаштування: зростаючі записи на кожному сервері (рис. 1.11). Звісно мати лише по одному чанку (одній гарячій точці) на шард не є реалістично, оскільки після підключення додаткового шарду виникне проблема: на новому шарді не відбуваються записи через відсутність гарячої точки для нього. Таким чином потрібно мати по декілька гарячих точок для кожного серверу, щоб був простір для росту. Але не потрібно робити їх занадто багато, адже це призведе до того, що записи будуть відбуватись випадковим чином (як при розподілі за хешовим ключем) [4].

Дану стратегію можна зобразити так: кожен чанк є стеком висхідних документів. На кожному шарді знаходиться декілька таких стеків, які збільшуються поки не досягнуть межі поділу. Після поділу один з чанків буде гарячою точкою, а інший буде «мертвим» (він більше ніколи не буде рости). Якщо стеки рівномірно розміщені по всьому кластері, то записи також будуть рівномірно розподілені [4].

Переваги даної стратегії:

- підвищена ефективність запису;
- якщо другим полем виступає поле з унікальними значеннями, то відсутня ймовірність появи «джамбо-чанків» та розміри «мертвих» чанків рівні.

Недоліки даної стратегії:

- неповністю вирішує проблему «джамбо-чанків»;
- неякісна (небезпечна) можливість встановлення навантаження на більш-/менш-потужні сервери: можливо виділити шарду більше/менше гарячих точок, але завжди існує ймовірність, що балансувальник перемістить гарячу точку на інший сервер.

1.2. Обґрунтування теми магістерської дисертації

Розглянувши існуючі стратегії розподілу даних було виявлено, що сучасні підходи не у всіх випадках гарантують розподіл який очікується користувачем. Однією з причин цього є те, що розподіл даних за кількістю чанків не гарантує необхідного розподілу. А це в свою чергу впливає на продуктивність роботи системи у цілому, оскільки навантаження на сервери може не відповідати їх нормам.

У деяких підходах особливо виділяється проблема «джамбо-чанків», але суттєвих дій для її уникнення в самих підходах не прийнято, хоча присутні попередження. На даний момент є два підходи вирішення проблеми з «джамбо-чанками»:

1) зменшити ймовірність їх появи шляхом використання складеного ключа шардингу – цей підхід не вирішує повністю поставлену проблему. Також розробники MongoDB рекомендують використовувати мінімальну кількість полів в якості ключа шардингу, що суперечить даному підходу;
2) при розподілі враховувати можливість наявності «джамбо-чанків», переміщуючи їх на відповідні їм шарди, шляхом тимчасової зміни максимально допустимого розміру чанків.

Тому метою даної роботи стало створення нового підходу, який зможе вирішити проблему «джамбо-чанків», також даний підхід має надавати можливість встановлювати відсоткове навантаження для кожного шарду в кластері, адже це надасть можливість підвищити продуктивність роботи систем, що використовують СУБД MongoDB.

2. ОПИС ЗАПРОПОНОВАНОГО ПІДХОДУ

Оскільки на сьогоднішній день не реалізовано підходів щодо врахування наявності «джамбо-чанків», доцільним є його розробка. Запропонований підхід має вирішувати дві поставлені задачі, а саме:

- при розподілі даних враховувати можливість наявності «джамбо-чанків» (тобто даний підхід має їх опрацьовувати, а не ігнорувати);
- можливість налаштування розподілу даних відповідно до допустимих відсоткових навантажень вузлів кластера (більш потужним серверам – більше навантаження та навпаки).

Для вирішення першої задачі є два варіанти:

1) зменшити ймовірність їх появи шляхом використання складеного ключа шардингу – цей підхід не вирішує повністю поставлену проблему. Також розробники MongoDB рекомендують використовувати мінімальну кількість полів в якості ключа шардингу, що суперечить даному підходу;
2) при розподілі враховувати можливість наявності «джамбо-чанків», та реалізувати їх переміщення на відповідні їм шарди, шляхом тимчасової зміни максимально допустимого розміру чанків.

Як зрозуміло з першого варіанту – він не повністю вирішує поставлену задачу, також під час роботи системи небажано змінювати ключ шардингу, тому доцільно використовувати другий варіант. Перевагою другого варіанту є те, що він не заперечує використанню складеного ключа для зменшення ймовірності появи «джамбо-чанків», а лише надає можливість їх переміщення на відповідні їм сервери.

Другу задачу можна вирішити в два способи:

1) відключити балансувальник та власноруч переміщувати всі чанки на відповідні їм сервери (відповідно до виставлених відсоткових навантажень);
2) використовувати зони.

Перший спосіб не є оптимальним, якщо переміщення буде відбуватись лише в одному потоці. Адже звичайні чанки балансувальник може переміщувати в декількох потоках (див. підрозділ 2.1), тому час затрачений на розміщення чанків буде меншим. Але мінусом використання балансувальника в даному випадку буде те, що він завжди буде пробувати розмістити майже рівну кількість чанків на кожному сервері (це у випадку без використання зон), що противорічить поставленій задачі. Таким чином використовувати перший спосіб буде доцільним лише при реалізації багато-потокового переміщення чанків.

Другий спосіб більше підходить для вирішення поставленої задачі, тому що:

- однією з причин створення зон є необхідність встановлення більшого/меншого навантаження на більш/менш-потужні сервери (див. підрозділ 1.3);
- потрібно лише реалізувати переміщення «джамбо-чанків», а за переміщення звичайних чанків може відповідати балансувальник (оскільки при використанні зон не виникатиме проблема описана для першого способу).

Таким чином алгоритм підходу щонайменше має мати наступні кроки виконання:

1. Розрахувати кількість чанків для кожного шарду.
    1.1. Розрахувати, який відсоток даних має бути на кожному шарді.
    1.2. З колекції «chunks» бази даних «config» отримати та відсортувати за зростанням значення ключа дані про чанки в заданій колекції.
    1.3. Дізнатись загальний розмір заданої колекції.
    1.4. Розрахувати кількість чанків для кожного шарду так, щоб була відповідність до розрахованих відсотків.
2. Знайти відповідні (нові) мінімальні та максимальні значення ключів для кожної зони та змінити існуючі (старі) значення ключів зон.

3. Відключити балансувальник.
4. Перемістити «джамбо-чанки» на відповідні їм шарди наступним чином.
   4.1. Змінити максимально допустимий розмір чанка на такий, щоб після зміни, існуючі «джамбо-чанки» вважались звичайними чанками.
   4.2. Перемісти всі «джамбо-чанки».
   4.3. Повернути максимально допустимий розмір чанків до початкового значення.
5. Включити балансувальник, щоб він перемістив всі інші чанки.

Крок 1.1 має виконуватись коли користувач вирішив встановити власні відсоткові значення для серверів, у інших же випадках відсоток даних для кожного шарду має розраховуватись за наступною формулою:

$$X = \frac{1}{n} \times 100\% ,$$

де X – це відсоток даних для кожного серверу, а *n* – це кількість серверів між яким розподіляється задана колекція.

Колекція «chunks» бази даних «config» зберігає дані про всі існуючі чанки в системі, тому для знаходження інформації про чанки в заданій колекції потрібно використати дану команду пошуку:

> db.chunks.find({ "ns" : "НБД.НК" }) [7]

Після чого знайдені записи потрібно відсортувати за зростанням нижньої або верхньої границі значень ключа в кожному шарді – це необхідно зробити, оскільки зона може мати лише один діапазон значень та діапазон не може перетинатись з іншими діапазонами зон для даної колекції. Для виконання даного сортування використовуються одна з наступних команд:

«отриманий список документів».sort({"min" : 1}) [7] – сортування за нижньою границею;

«отриманий список документів».sort({"max" : 1}) [7] – сортування за верхньою границею.

У документів колекції «chunks» присутні поля «min» та «max» які зберігають в собі значення нижньої та верхньої границі.

Для виконання кроку 1.3 потрібно використати команду *db.НК.dataSize()* [7], яка повертає розмір колекції в байтах.

Крок 1.4 виконується в декілька етапів:

1) розрахувати рекомендований розмір для серверу за формулою:

$$Size = \frac{dataSize \times X}{100},$$

де *dataSize* – це повний розмір колекції, *X* - відсоток даних для даного серверу, а *Size* – це рекомендований розмір.

2) Створити змінні currentSize та chunksCount присвоївши їм 0;
3) Запустити цикл проходження по списку документів отриманому під час кроку 1.2.
4) Отримуючи документ перевіряти чи сума його розміру (chunkSize; отримується за допомогою використання команди dataSize [7]) та значення в *currentSize* буде більшою ніж *Size*. Якщо «Ні», то змінну *currentSize* збільшити на розмір чанку, а *chunksCount* збільшити на одиницю. Якщо «Так», то виконати наступну перевірку:

   ((currentSize + chunkSize) – Size) > (Size – currentSize)

   Дана дія перевірить при якому результаті *currentSize* буде ближчим до рекомендованого розміру (*Size*).

   Якщо результатом перевірки є «true», то нічого не робити (з даного чанку має початись розрахунок для наступного серверу).

   Якщо результатом перевірки є «false», то змінну *currentSize* збільшити на розмір чанку, а *chunksCount* збільшити на одиницю (розрахунок для наступного серверу має початись з наступного чанку в списку).
5) Вийти з циклу.

6) Зберегти значення *chunksCount* для його подальшого використання.

У кроці 2 потрібно проходячи по списку документів отриманому під час кроку 1.2 знайти нові значення нижніх та верхніх границь зон, після чого замінити ними старі.

Крок 3 є бажаним для виконання, оскільки в подальшому можливо будуть виконуватись переміщення «джамбо-чанків» і для того, щоб не виникали ситуації, коли для переміщення потрібно чекати доки закінчаться переміщення ініціалізовані (запущені) балансувальником.

Четвертий крок як і вказано виконується в 3 етапи. Для зміни максимально допустимого розміру чанків потрібно підключитись до бази даних «config» (командою: *db = db.getSisterDB("config")*) та використати наступну команду:

> db.settings.save({ _id:"chunksize", value: «розмір (в Мб.)»})

Дана команда створить або перезапише (якщо існує) документ в якому будуть зберігатись налаштування максимально допустимого розміру чанків.

Останнім кроком є запуск балансувальника. Даний крок є обов’язковим, якщо був виконаний крок №3, оскільки без його виконання всі звичайні чанки залишаться там де вони і були.

Через особливості роботи авто-поділу чанків (див. підрозділ 2.2), рекомендується додати «Крок 0» в якому має виконуватись поділ всіх чанків які досягли максимально допустимого розміру (винятками є «джамбо-чанки»). Причиною даної рекомендації стали ситуації в яких, при розрахунку кількості чанків на сервер, розмір останнього чанка, який є спірним (виникла ситуація з додатковою перевіркою), досяг максимально допустимого розміру та його можна поділити. У таких випадках, якби відбувся поділ цього значення, то розрахований розмір даних на сервер мав би ближче значення до рекомендованого розміру. Тобто точність розрахунків збільшиться.

Наприклад, при розрахунку розміру даних на певний сервер сталась спірна ситуація, до розрахункового розміру залишилось 0.5 Мб, але наступний чанк має розмір 1.6 Мб та його можна поділити, максимально допустимий розмір чанку – 1 Мб. При роботі алгоритму (без попереднього поділу) відбудеться перевірка:

0.5 ? 1.1 = (1.6 – 0.5)

Після даної перевірки буде встановлено, що кращим варіантом буде не включати даний чанк до зони, оскільки так різниця між рекомендованим та розрахованим розмірами буде становити лише 0.5 Мб, а не 1.1 Мб.

Якщо в роботу алгоритму включити попередній поділ чанків, то спірний чанк буде розбитий на два чанки з майже або однаковим розміром (наприклад, після поділу було отримано два чанки з розмірами в 0.8 Мб). То в такому випадку, перевірка матиме наступний вигляд:

0.5 ? 0.3 = (0.8 – 0.5)

У цьому випадку, першу частину чанку буде включено до зони, а друга частина буде включена в іншу зону. Також різниця між рекомендованим та розрахованим розмірами буде меншою, що підвищить точність роботи алгоритму.

Для правильної роботи даного підходу потрібне виконання наступних умов:

1. Існування власної зони у кожного шарда.
2. Наявність даних про шарди та колекції для розподілу.

Перша умова є обов’язковою для виконання, оскільки без зон даний алгоритм виконувати не зможе.

Другу умову можна реалізувати двома способами:

1) створити змінні в яких будуть зберігатись дана інформація;
2) створити окрему колекцію в якій документи будуть містити необхідну інформацію.

Перший спосіб не потребує ніяких дій, але зміна налаштувань буде можлива лише на сервері, де була об’явлені дані змінні.

Другий спосіб потребує додаткових операцій зчитування інформації з документів (наприклад, за використанням алгоритму зчитування налаштувань (див. підрозділ 3.1)), але зміна налаштувань буде можливою на будь-якому сервері.

Саму ж необхідну інформацію можна поділити на два вити: ту, що можна згенерувати (отримати програмно) та ту, що має надати користувач.

Про шарди мають надаватись наступні дані:

1) назва шарду (можна отримати з колекції «shards» бази даних «config»);
2) бали шарду (має надати користувач, або за замовчуванням призначати 1);
3) назва зони до якої даний шард підключено (можна отримати з колекції «shards» бази даних «config»).

Інформація про всі підключені шарди зберігається в колекції «shards» бази даних «config». У кожному документі даної колекції є поле «_id» в якому зберігається назва шарду, поле «host» в якому зберігається повна назва шарду (крім назви включена інформація про самі сервери, які містить шард, його репліки), поле «state» у якому збережено значення стану шарда та поле «tags» яке зберігає всі назви зон у які входить даний шард.

Бали шарду використовуються при розрахунку відсоткового навантаження шарду, тобто чим їх більше – тим більше даних буде виділено шарду.

Про колекції мають надаватись наступні дані:

1) назва колекції (можна отримати з колекції «collections» бази даних «config»);
2) мін. значення поля-ключа (можна отримати програмним способом);

3) макс. значення поля-ключа (можна отримати програмним способом);
4) назва бази даних у якій знаходиться колекція (можна отримати з колекції «collections» бази даних «config»);
5) назва поля-ключа (має надати користувач).

Інформація про колекції допущені до розподілу, зберігається в колекції «collections» бази даних «config». У кожному документі даної колекції є поле «ns» в якому зберігається назва шардингової колекції та в якій базі даних вона знаходиться. Також там є й інформація про ключ шардингу, але отримати назви полів-ключів звідти неможливо, якщо бути точнішим, то можливе отримання самого ключа щардингу з усіма його полями та їх значеннями, але для їх зміни, що й буде потрібно в деяких частинах програмного коду, потрібно зазначити назви даних полів (для звернення). Тому користувач має надавати назву поля-ключа.

Мінімальні та максимальні значення ключа можна отримати безпосередньо з самої колекції використовуючи алгоритм пошуку мінімального/максимального значення поля-ключа колекції (див. підрозділ 3.2).

2.1. Процес балансування даних в MongoDB

Балансувальник MongoDB є внутрішнім процесом СУБД, який працює в окремому потоці та відповідає за моніторинг кількості чанків для кожного шарду. Коли кількість чанків на серверах досягають визначених границь переміщення (див. пункт 2.1.4), балансувальник намагається автоматично перемістити чанки між серверами та досягти однакової кількості чанків для кожного серверу [9].

Процедура балансування для шардингового кластеру є повністю прозорою для користувача та додатків, хоча підчас переміщення може спостерігатись понижена продуктивність роботи системи [9].

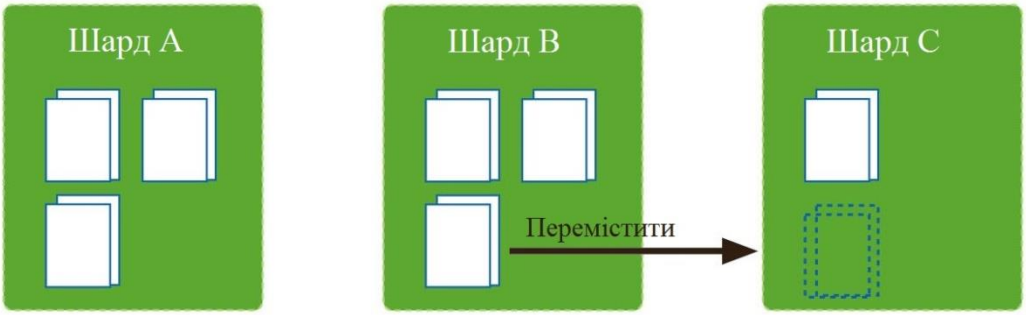

Рисунок 2.1 – Демонстрація роботи балансувальника

На рис. 2.1 продемонстровано роботу балансувальника, у ситуації, коли кластер складається з трьох шардів та на двох шардах знаходиться по три чанки, а на останньому – один чанк. Балансувальником буде визначено, що потрібно на третій шард (з одним чанком) перемістити один чанк для більш рівномірного розподілу [9].

2.1.1. Балансувальник кластеру

Балансувальник відповідає за рівномірний перерозподіл чанків шардингової колекції між усіма підключеними серверами. За замовчуванням процес балансування завжди включений [9].

Для вирішення проблеми нерівномірного розподілу чанків, балансувальник переміщує чанки з більш навантажених шардів до менш навантажених. Процес балансування, переміщення чанків, продовжується доки розподіл не стане рівномірним (більш детально процедура переміщення чанків описана в пункті 2.1.3) [9].

Сам процес переміщення чанків супроводжує додаткові витрати з точки зору пропускних спроможностей та навантажень, що в свою чергу може впливати на продуктивність роботи бази даних та системи, яка використовує її. Тому для зменшення цього впливу розробниками MongoDB були прийняті такі правила:

- шард може виконувати паралельне переміщення чанків, кількість можливих одночасних переміщень розраховується наступним

чином: *n/2* (округлення до меншого цілого), де *n* – кількість підключених шардів [9];

- починати процес балансування лише у випадках, коли розподілені дані досягають границь переміщень (див. пункт 2.1.4) [9].

Інколи в системах, що використовують недостатньо потужні машини для її підтримання, є необхідність відключити процес балансування для зменшення навантаження на неї. Для припинення роботи балансувальника для всіх існуючих в системі колекцій використовується команда:

> sh.stopBalancer(timeout, interval)

, де *timeout* – часовий ліміт відключення колекції (за замовчуванням рівний 60 000 мілісекунд), а *interval* – це інтервал в якому потрібно перевірити припинив свою роботу балансувальник (вказується в мілісекундах) [7].

Якщо необхідно відключити процес балансування лише для певної колекції, то використовується наступна команда:

> sh.disableBalancing(namespace)

, де *namespace* вказує назву бази даних в якій знаходиться колекція та саму назву колекції в форматі – "НБД.НК" [7].

2.1.2. Підключення та відключення шарду до кластеру

Підключення нового шарду призводить до незбалансованого розподілу, оскільки на новому шарді відсутні будь-які чанки. Навіть якщо MongoDB почне відразу переміщувати на нього дані, має пройти певний час перед тим як дані знову будуть збалансовані [9].

Відключення шарду також призводить до аналогічного незбалансованого розподілу, оскільки дані розташовані на шарді мають бути перерозподілені по всьому кластеру. Як і у випадку підключення шарду, MongoDB почне переміщувати дані, але це займе час, щоб знову дані були збалансовані. Також під час даного процесу небажано відключати сервери [9].

Також після відключення шарду, дані не будуть обов’язково збалансовані, тому знову буде запущений процес балансування [9].

2.1.3. Процедура переміщення чанків

Переміщення чанків відбувається за наступним алгоритмом [9]:

1. Балансувальник відправляє запит на переміщення чанка (за допомогою команди *moveChunk* [7]).
2. Шард-відправник (з якого переміщуються дані) починає процес переміщення даних за допомогою внутрішньої команди *moveChunk* [7]. Під час процесу переміщення, операції, які звертаються до даних в чанку, опрацьовуються шардом-відправником.
3. Шард-отримувач (який приймає чанк), створює нові індекси (посилання), які необхідні для правильного функціонування системи після переміщення.
4. Шард-отримувач починає запитувати документи з чанку та починає отримувати їх копії.
5. Після отримання всіх документів, шард-отримувач запускає процес синхронізації для перевірки чи всі документи були скопійовані (є можливість, що підчас переміщення в чанк були записані нові документи).
6. Після повної синхронізації, шард-відправник підключається до бази даних «config» та обновити метадані кластеру з новим місцезнаходженням чанку.
7. Після обновлення метаданих та закінчення відкритих запитів до даних в чанків, шард-відправник видаляє копію чанка.

Даний алгоритм переміщення забезпечує узгодженість та максимальну доступність даних під час балансування [9].

Також потрібно зазначити, що вразі необхідності запустити додаткові переміщення з шарда-відправника, балансувальник може

запустити новий процес переміщення не очікуючи закінчення вже запущеного (див. пункт 2.1.5) [9].

2.1.4. Границі переміщень

Для зменшення впливу процесу балансування в кластері на систему, балансувальник починає перерозподіляти чанки між сервери лише у випадках досягнення певних границь переміщення. У якості яких виступає значення різниці мінімальної та максимальної кількості чанків на шардах. Границі переміщень наведені в таблиці 2.1 [9].

Таблиця 2.1 - Границі переміщень

Кількість чанків	Границі переміщення
Менше 20	2
20 - 79	4
Більше 80	8

Балансувальник закінчує працювати коли різниця між кількістю чанків менше двох або коли переміщення неуспішні [9].

2.1.5. Асинхронне переміщення чанків

Для переміщення декількох чанків з одного шарду, балансувальник переміщує їх послідовно, але він не очікує закінчення вже запущеного процесу переміщення, а створює чергу переміщень [9].

Створення черги переміщень надає шардам більш швидко переміщувати дані у випадках їх нерівномірного розподілу, наприклад: при виконання запису даних без первинного їх поділу, при підключенні нових шардів [9].

Це також впливає на роботу команди *moveChunk* та записи переміщень, які використовують *moveChunk*, можуть виконуватись швидше [9].

У деяких випадках, фази видалення можуть продовжуватись довше. Якщо багато фаз видалення запущені, але незавершені, поломка

первинного (primary) серверу з набору реплік може призвести до «осиротіння» даних з багатьох процесів переміщень [9].

2.1.6. Максимальна кількість документів для чанка

MongoDB не може переміщувати чанки в яких кількість документів більше в 1.3 рази ніж результат поділу максимально допустимого розміру чанків на середній розмір документів в колекції. Середній розмір документу береться з поля «avgObjSize» результату роботи команди *db.НК.stats()* [9].

2.2. Авто-поділ (Autosplit)

mongos-сервери слідкують за тим скільки документів вони записують в кожен чанк. У ситуаціях коли кількість записів досягає певного значення, відбувається перевірка на необхідність поділу (як зображено на рис. 2.2 та 2.3). Якщо поділ непотрібен, то *mongos*-сервер лише обновить метадані на конфігураційних серверах [4].

Поділ чанків – це лише зміна метаданих (не відбувається переміщення даних). Нові записи створюються на конфігураційних серверах та відбувається модифікація існуючих: значення діапазонів змінюються на актуальні. Після завершення даного процесу, *mongos*-сервери заново визначають вказівники на дані в кластері та створюють нові вказівники для новостворених чанків [4].

Коли *mongos*-сервер посилає запит шарду на перевірку потрібності поділу чанків, шард робить приблизні розрахунки розмірів чанків. Якщо на шарді наявні чанки з розмірами перевищуючими максимально допустимі, то він знаходить їх можливі точки поділу та відправляє цю інформацію *mongos*-серверу (рис. 2.4) [4].

Хоча можуть виникати ситуації, коли шард не знайде точки поділу, навіть для великих за розмірами чанків, оскільки їх обмежена кількість для всієї колекції. Будь-які два документи з однаковими значеннями ключа

мають знаходитись в одному чанку, тому чанки можуть буди поділені лише в місцях зміни значень ключа шардингу. Наприклад, якщо ключем є поле «age», то наступний чанк може буде поділений лише у вказаних місцях:

{"age": 13, "username": "ian"}
{"age": 13, "username": "randolph"}
---------- *// місце поділу*
{"age": 14, "username": "randolph"}
{"age": 14, "username": "eric"}
{"age": 14, "username": "hari"}
{"age": 14, "username": "mathias"}
---------- *// місце поділу*
{"age": 15, "username": "greg"}
{"age": 15, "username": "andrew"}

Mongos-сервер необов'язково поділить чанк у всіх можливих місцях поділу, але вони є варіантами з яких буде відбуватись вибір [4].

Наприклад, якщо в чанку зберігаються наступні дані, то його неможливо поділити (поки додаток не почне використовувати десяткові числи для поля «age»):

{"age": 12, "username": "kevin"}
{"age": 12, "username": "spencer"}
{"age": 12, "username": "alberto"}
{"age": 12, "username": "tad"}

Тому важливо мати велику різноманітність значень ключа шардингу.

У разі, якщо один з конфігураційних серверів вимкнений, коли mongos-сервер пробує поділити чанки, то mongos-сервер не зможе обновити метадані (рис. 2.5). Усі конфігураційні сервери мають працювати та мати можливість підключення до них, щоб поділ відбувся. Якщо mongos-сервер буде продовжувати отримувати запити на запис, то він буде пробувати поділити чанки, але це буде безуспішно. До тих пір поки конфігураційні сервери не будуть правильно працювати, поділ не відбудеться та всі спроби поділу будуть сповільнювати роботу як mongos-серверу так і шардів (відбувається повторення процесу зображеного на рис. 2.2 через процес зображений на рис. 2.5 для кожного вхідного запису).

Неуспішний процес повторних спроб mongos-серверу поділити чанки називається «штормовим поділом». Єдиним способом запобігти появі «штормового поділу» є надання стовідсоткової можливості підключення до конфігураційних серверів та забезпечення їх правильної роботи. Також щоб скинути значення лічильника записів на значення по замовчуванню, можна перезапустити *mongos*-сервер (таким чином він більше не буде знаходитись на границі поділу) [4].

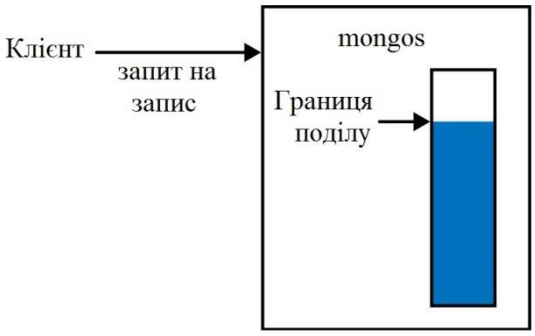

Рисунок 2.2 – При записі документів в чанк, mongos-сервер перевіряє границі поділу для чанка.

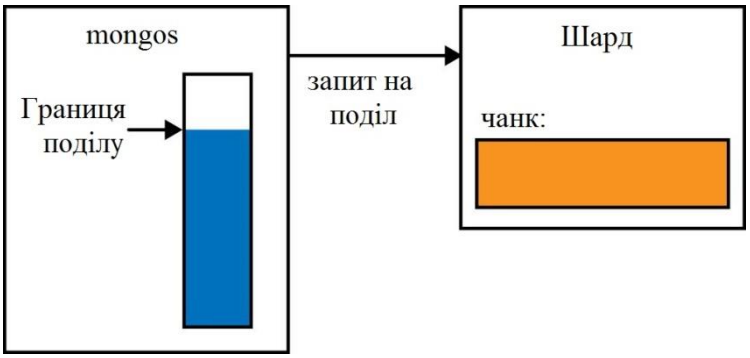

Рисунок 2.3 – При досягнені границі поділу, mongos-сервер відправляє запит шарду запит для пошуку точок поділу.

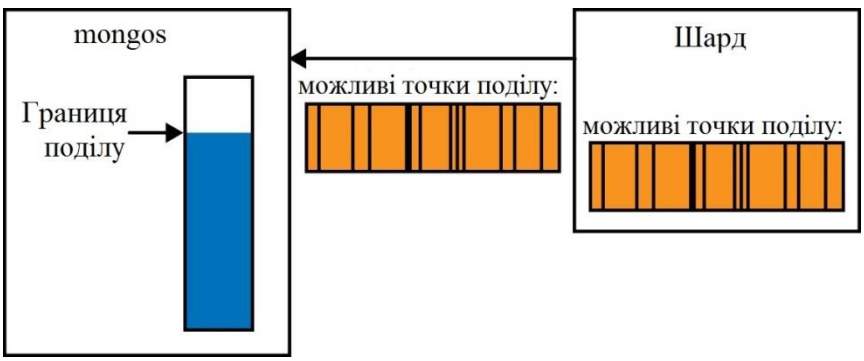

Рисунок 2.4 – Шард розраховує точки поділу та результат відправляє mongos-серверу.

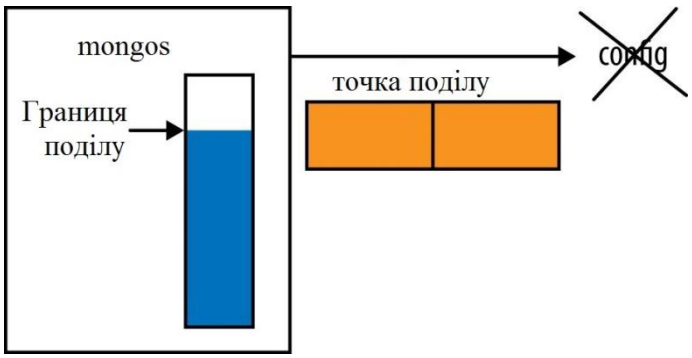

Рисунок 2.5 – mongos-сервер вибрав точку поділу, але не може підключитись до конфігураційних серверів.

Іншою проблемою є те, що *mongos*-сервер може не зрозуміти, що йому потрібно поділити великий чанк. У MongoDB немає ніякого загального лічильника, який би вказував наскільки великі за розміром чанки в кластері. Кожен *mongos*-сервер розраховує чи записи які він опрацював доводять його до границі поділу (рис. 2.6). Це означає, що в разі, коли процеси в *mongos*-сервері часто прискорюються та сповільнюються, може виникнути ситуація, коли не буде досягнена необхідна кількість записів для входження в границю поділу перед його вимкненням. Тобто розмір існуючих чанків буде постійно збільшуватись (рис. 2.7) [5].

Першим способом запобігти цього є зменшити кількість mongos-серверів. Залишати mongos-сервери працюючими, коли це я можливим, та вимикати їх, коли ні. Проте користувач може знайти дуже затратним залишати невикористані mongos-сервери. У такому разі, іншим способом отримати більше процесів поділу є зменшити максимально допустимий розмір чанків – що призведе до зменшення критеріїв досягнення границі поділу [4].

MongoDB надає можливість відключити авто-поділ чанків, для цього при запуску *mongos*-серверу потрібно дописати *«--nosplit»* або використати команду *sh.disableAutoSpli()* [4].

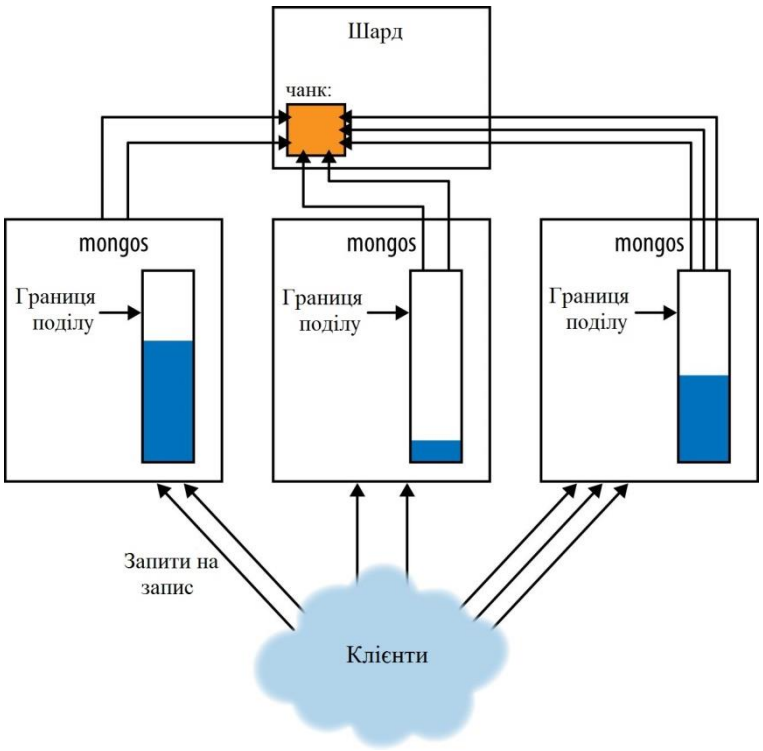

Рисунок 2.6 - При опрацюванні запитів на запис, значення лічильників mongos-серверів зростають.

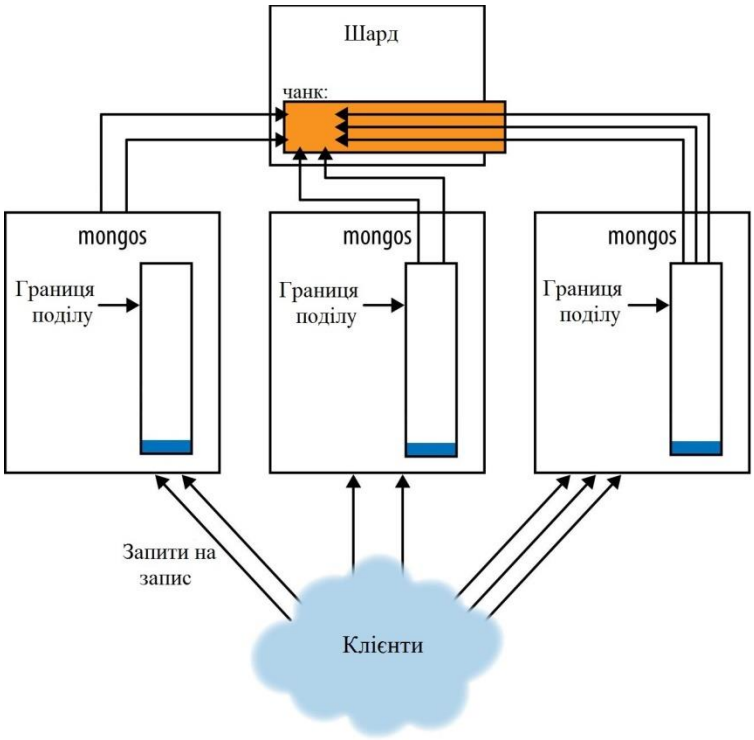

Рисунок 2.7 - У разі частого перезапуску mongos-серверів, вони можуть не досягти границь поділу, що призведе до постійного збільшення розміру чанків.

3. АЛГОРИТМІЧНІ ОСОБЛИВОСТІ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

У даному розділі розглядаються особливості програмної реалізації запропонованого підходу та алгоритмічні особливості розроблених програмних засобів.

Загалом розглядаються наступні алгоритми (у дужках вказані функції, які реалізують їх програмно; лістинг функцій надано в додатку 1):

- зчитування налаштувань (readShard, readCollections);
- пошуку мінімального/максимального значення поля-ключа в колекцій (findMin, findMax);
- перевірки чанка (checkChunk);
- розрахунку відсоткового навантаження (CountPercentages);
- поділу чанків (ChunkSpliter);
- розрахунку кількості даних для шардів, за чанками (ChunksCount);
- розрахунку кількості даних для шардів, за пам’яттю (MemoryCount);
- зміни границь зон (ZoneBalancer);
- переміщення «джамбо-чанків» (JumboMover);
- створення зон (setZones);
- видалення зон (removeZones);
- додання документів до колекцій (insertFiles);
- перевірки використаної пам’яті (MemoryCheck).

Потрібно зазначити, що для реалізації запропонованого підходу потрібно використати наступні алгоритми: алгоритм поділу чанків, алгоритм розрахунку кількості даних для шардів (за пам’яттю), алгоритм зміни границь зон та алгоритм переміщення «джамбо-чанків».

3.1. Алгоритм зчитування налаштувань

Даний алгоритм відповідає за зчитування інформації про шарди/колекції збереженої в певній колекції (наприклад, колекції «info»

бази даних «config»). Результатом його роботи є масив у якому зберігаються дані налаштувань.

Алгоритм має наступні кроки:

1. Підключення до бази даних де знаходиться колекція з даними;
2. У колекції виконати пошук документів які мають поле «shard»/«collection» (наприклад, якщо документ має поле «shard», то це означає, що він зберігає інформацію про певний шард);
3. Проходячись по знайденим документам записати дані в масив;
4. У якості результату роботи повернути отриманий масив з даними.

Для правильної роботи даного алгоритму йому не потрібно надавати додаткові дані, а результат його роботи буде залежати повністю від користувача.

3.2. Алгоритм пошуку мінімального/максимального значення поля-ключа в колекції

Даний алгоритм відповідає за знаходження мінімального/ максимального значення поля-ключа в колекції. Для правильної роботи даного алгоритму йому потрібно вказати по якому полю буде виконуватись пошук та в якій колекції.

Алгоритм має наступні кроки:

1. Підключення до бази даних в якій знаходиться колекція;
2. Сортування документів колекції за зростанням/зменшенням значень поля-ключів;
3. Взяти перший документ (у ньому й буде зберігатись необхідне значення);
4. У якості результату роботи повернути шукане значення.

3.3. Алгоритм перевірки чанка

Даний алгоритм відповідає за перевірку чанка на наявність документів з різними значеннями поля-ключа. Для його правильної

роботи, йому необхідно надати інформацію про чанк, який потрібно перевірити, та інформацію про колекцію, до якої він належить.

Результатом роботи алгоритму є логічне значення «true» – в разі якщо чанк зберігає максимум одне значення поля-ключа, та «false» – в усіх інших випадках.

Алгоритм має наступні кроки:

1. Підключення до бази даних в якій знаходиться чанк;
2. Створення змінної лічильника та присвоєння їй значення 0:
3. Створення змінної «tmp» в яку зберегти значення нижньої границі чанку;
4. Дізнатись кількість існуючих документів у яких значення поля-ключа рівні значенню в змінній «tmp». Якщо кількість документів більша за 0, то значення лічильника збільшити на 1;
5. Якщо значення лічильника рівне або більше 2, то припинити роботу алгоритму, в якості результату повернути «false». Якщо ж значення лічильника менше 2, то знайти наступне значення поля-ключа в чанку, зберегти його в змінну «tmp» та повернутись до 4 кроку. Якщо наступне значення поля-ключа в чанку відсутнє, то припинити роботу алгоритму, в якості результату повернути «true».

3.4. Алгоритм розрахунку відсоткового навантаження

Даний алгоритм розраховує відсоткове навантаження для кожного шарда. Для його правильної роботи йому потрібно отримати дані про шарди (це реалізується за допомогою використання алгоритму зчитування налаштувань). Після чого використовуючи дані про шарди та формулу наведену в описі першого кроку запропонованого підходу (див. розділ 2) розрахувати відсоткове навантаження.

3.5. Алгоритм поділу чанків

Даний алгоритм відповідає за поділ чанків розмір яких перевищив максимально допустимий, при цьому дані чанки не мають бути «джамбо-чанками».

Даному алгоритму, перед початком його роботи, необхідно надати інформацію про колекцію в якій буде відбуватись поділ (можна використати результат роботи алгоритму зчитування налаштувань) та вказати максимально допустимий розмір чанку (зазвичай зберігається в колекції «settings» бази даних «config», а його відсутність там означає, що його значення рівне 64 Мб).

Алгоритм має наступні кроки:

1. Підключення до бази даних «config»;
2. Отримання списку всіх чанків вказаної колекції з колекції «chunks»;
3. Для кожного чанка в отриманому списку виконати дві перевірки:
   1) чи перевищує розмір чанка максимально допустимий (використовуючи команду *dataSize* [7]);
   2) чи знаходиться в чанку декілька значень поля-ключа (для цього можна використати алгоритм перевірки чанка з підрозділу 3.3).

      Якщо розмір чанку перевищує максимально допустимий та результат другої перевірки є «false», то виконати поділ чанки за допомогою команди *sh.splitFind()* [7]. Дана команда ділить чанк на дві приблизно рівні частини;
4. Вивести користувачу повідомлення про поділені чанки.

Потрібно зазначити: оскільки команді *dataSize* потрібно надавати дійсні значення поля-ключа, то у випадку використання розподілу за хешовим ключем, даний алгоритм не буде працювати.

3.6. Алгоритм розрахунку кількості даних для шардів (за чанками)

Даний алгоритм розраховує кількість чанків для кожного шарду не враховуючи їх розмір. Для роботи даного алгоритму потрібно вказати

загальну кількість чанків в колекції та відсоткове навантаження для кожного шарду (отримується з результату виконання алгоритму відсоткового навантаження).

Алгоритм працює наступним чином:

1. Створити змінну «percent» в яку записати скільки чанків складає 1% від всіх чанків, тобто потрібно загальну кількість чанків поділити на 100. Значення в змінній може бути нецілим числом;
2. Розрахувати кількість чанків для всіх шардів крім останнього в списку за наступною формулою:

$$chunks = percent \times shardPercentage,$$

де *percent* – значення змінної, а *shardPercentage* – це відсоткове навантаження певного шарду. Після цього потрібно округлити кількість чанків до цілого значення (в більшу сторону);
3. Розрахувати кількість чанків для останнього шарду. Для цього потрібно від загальної кількості чанків відняти всі вже розраховані кількості чанків;
4. У якості результату повернути розраховану кількість чанків для кожного шарду.

3.7. Алгоритм розрахунку кількості даних для шардів (за пам’яттю)

Даний алгоритм розраховує кількість чанків для кожного шарду враховуючи їх розмір. Для роботи даного алгоритму потрібно вказати інформація про колекцію для якої будуть вестись розрахунки та відсоткове навантаження для кожного шарду (отримується з результату виконання алгоритму відсоткового навантаження).

Етапи роботи даного алгоритму було розглянуто в розділі 2 при детальному описі кроку 1.4 запропонованого підходу.

Потрібно зазначити: оскільки команді *dataSize* потрібно надавати дійсні значення поля-ключа, то у випадку використання розподілу за хешовим ключем, даний алгоритм не буде працювати.

3.8. Алгоритм зміни границь зон

Даний алгоритм відповідає за зміну значень нижніх та верхніх границь зон. Для його роботи потрібно надати інформацію про те скільки чанків має стати на кожному шарді (отримується за допомогою використання одного з алгоритмів розрахунку кількості даних для шардів), інформацію про колекцію границі зон якої мають бути змінені та інформацію про шарди. Інформація про колекцію та шарди отримується з результату роботи алгоритму зчитування налаштувань.

Алгоритм працює наступним чином:

1. Підключення до бази даних «config»;
2. Отримання списку всіх існуючих чанків в колекції (дана інформація зберігається в колекції «chunks»);
3. Використовуючи розраховані кількості чанків для кожного шарду, пройтись по списку чанків та знайти нові значення границь зон;
4. У колекції «tags», обновити документи, що описують необхідні зони, новими значеннями границь.

Потрібно зазначити, що функція ZoneBalancer включає в себе реалізацію даного алгоритму, але крім цього вона ще реалізує запропонований підхід.

3.9. Алгоритм переміщення «джамбо-чанків»

Даний алгоритм відповідає за переміщення «джамбо-чанків» на відповідні їм шарди. Йому для правильної роботи необхідна інформація про колекцію над чанками якої він буде проводити дії, значення границь зон для даної колекції та інформація про шарди яким відповідають зони.

Алгоритм працює наступним чином:

1. Підключення до бази даних «config»;
2. Отримання списку всіх існуючих чанків в колекції (дана інформація зберігається в колекції «chunks»);

3. Збільшити максимально допустимий розмір чанків так, щоб усі чанки «джамбо-чанки» після зміни вважались звичайними чанками (наприклад, збільшити розмір у 1000 раз від початкового);
4. Проходячись по списку чанків виконувати наступні перевірки:
   1) чи є даний чанк «джамбо-чанком»;
   2) чи знаходиться він на відповідному для нього шарді (у виділеній йому зоні).

   Для першої перевірки потрібно перевірити його розмір за допомогою команди *dataSize* [7] та отримати результат виконання алгоритму перевірки чанка (див. підрозділ 3.3). Для другої перевірки потрібно використати інформацію про зони та сам чанк.

   Якщо чанк є «джамбо-чанком» та не знаходиться на відведеному йому шарді, то перемістити його у відповідну йому зону командою *moveChunk* [7].
5. Повернути максимально допустимий розмір до початкового значення.

Потрібно зазначити: оскільки команді *dataSize* потрібно надавати дійсні значення поля-ключа, то у випадку використання розподілу за хешовим ключем, даний алгоритм не буде працювати.

3.10. Алгоритм створення зон

Даний алгоритм створює зони для вказаних колекцій та підключає до них шарди. Для роботи даного алгоритму йому потрібно надати результат роботи алгоритму зчитування налаштувань.

Алгоритм має наступні кроки:

1. Підключення до бази даних «config»;
2. Генерація значень нижніх і верхніх границь зон;
3. Генерація документів, які мають описувати зони (у них є свій шаблон заповнення);

4. Додання згенерованих документів до колекції «tags» (це і виконає створення зон);
5. Підключити шарди до відповідних їм зонам використовуючи команду *sh.addShardToZone()* [7].

На даний момент програмний засіб, який реалізує даний алгоритм, може правильно створити зони лише у випадку коли поле-ключ є числового типу. Для того, щоб він міг створювати й для рядкового типу потрібно добавити йому можливість генерації рядкових значень для границь зон.

3.11. Алгоритм видалення зон

Даний алгоритм видаляє усі зони, які існують у вказаних колекціях, інформація про які отримується за допомогою алгоритму зчитування налаштувань. Для його правильної роботи йому необхідно надати інформацію про колекції та шарди (все отримується з результату виконання алгоритму зчитування налаштувань).

Алгоритм складається з таких кроків:

1. Підключення до бази даних «config»;
2. У колекції «tags» видалення документів пов’язаних з указаними колекціями. Це реалізується за допомогою команди *db.tags.remove()* [7].
3. Від’єднати шарди від зон командою *sh.removeShardFromZone()* [7] (інформацію про те якій зоні належить який шард має бути отримана з інформації про шарди).

3.12. Алгоритм додання документів до колекцій

Даний алгоритм відповідає за генерацію та додання документів до вказаних колекцій. Для його правильної роботи йому потрібно надати інформацію про колекції до яких мають бути додані документи, кількість документів та відсоток покриття діапазону значень поля-ключа для

вказаних колекцій. Відсоток покриття необхідний для реалізації тестування запропонованого підходу в різних ситуаціях.

Алгоритм має наступні кроки:

1. Запуск циклу для певної колекції;
2. Підключення до бази даних в якій знаходиться колекція;
3. Створення змінної лічильника ти присвоєння їй 0;
4. Генерація документа;
5. Додання документу до колекції, збільшення лічильника на 1;
6. Якщо значення лічильника менше кількості вказаних документів, то повернутись до кроку 4. Якщо ж значення лічильника рівне кількості вказаних файлів для додання, то перейти до наступної колекції (якщо така є, потрібно повернутись до кроку 1, в іншому випадку - припинити свою роботу).

3.13. Алгоритм перевірки використаної пам’яті

Даний алгоритм відповідає за перевірку використаної пам’яті для збереження даних колекції. Крім цього він надає можливість вказати користувачу які саме чанки є «джамбо-чанками», де вони розташовані та скільки їх всього на кожному шарді.

Для правильної роботи алгоритму необхідно надати інформацію про колекції та шарди (використовується алгоритм зчитування налаштувань).

Алгоритм складається з таких кроків:

1. Підключитись до бази даних «config»;
2. Створити змінну «maxChunkSize» та записати в неї значення максимально допустимого розміру чанків (береться з колекції «settings»);
3. Запустити циклу, який буде проходити по всіх вказаних колекціях;
4. З колекції «chunks» отримати список всіх чанків поточної колекції, відсортувати його за двома полями: шардом та мін./макс. значенням поля-ключа;

5. Створити змінну «size» та присвоїти їй 0, створити лічильник для «джамбо-чанків»;
6. Проходячись по списку чанків (поки чанки знаходяться на одному шарді):
   6.1. знаходити їх розмір (використовуючи команду *dataSize* [7]) та збільшувати змінну size на отримане значення;
   6.2. перевіряти чи є чанк «джамбо-чанком», якщо так, то лічильник збільшити на 1 та користувачу вивести повідомлення є вказуватиме, що чанк відповідає критеріям «джамбо-чанка».

   Якщо наступний чанк в списку знаходиться на іншому шарді, то вивести значення лічильника та змінної «size», після чого перейти до 5 кроку;
7. Після завершення проходження перейди до опрацювання наступної колекції в списку (перейти до кроку 3).

Потрібно зазначити: оскільки команді *dataSize* потрібно надавати дійсні значення поля-ключа, то у випадку використання розподілу за хешовим ключем, даний алгоритм не буде працювати.

4. ТЕСТУВАННЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

Кожне програмне забезпечення під час розробки проходить певні етапи, одним з яких є тестування. Цей етап є дуже важливим, адже він допомагає знайти помилки в розробленому програмному забезпечені або впевнитись в його правильності виконання. Таким чином потрібно провести тестування розроблених програмних засобів, а саме:

- функцій зчитування налаштувань;
- функцій пошуку мінімального/максимального значення поля-ключа в колекції;
- функції перевірки чанка;
- функції розрахунку відсоткового навантаження;
- функції розрахунку кількості даних для шардів (за чанками);
- запропонованого підходу, функцій додання документів до колекцій та розрахунку кількості даних для шардів (за пам’яттю);
- функцій створення та видалення зон.

Тестування проводиться на комп’ютері з такими характеристиками:

- процесор: Intel(R) Core(TM) i7-3632QM;
- частота процесора: 2.2ГГц – 3.2ГГц;
- оперативної пам’яті: 8 ГБ;
- операційна система: Window8.1 64-bit.

Також потрібно зазначити, що максимально допустимий розмір чанку буде встановлено 1 Мб.

4.1. Тестування функцій зчитування налаштувань

Тестування функції проводиться на наступних ситуаціях:

- відсутність жодної інформації в вказаній колекції;
- наявність лише по одному запису з налаштуваннями;
- наявність декількох записів.

У третій ситуації, зчитуючи дані про колекції, потрібно щоб надана інформація відповідала наступним умовам:

1) була інформація про колекції з однаковою назвою, але які знаходяться в різних базах даних;
2) була інформація про колекції з різними назвами, але які знаходяться в одній базі даних.

Дані ситуації описують всі можливі випадки з якими потрібно буде працювати даним функціям.

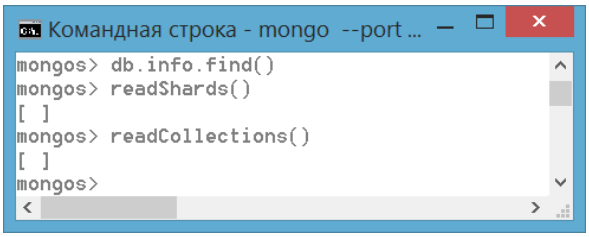

Рисунок 4.1 – Результат роботи функцій при першій ситуації

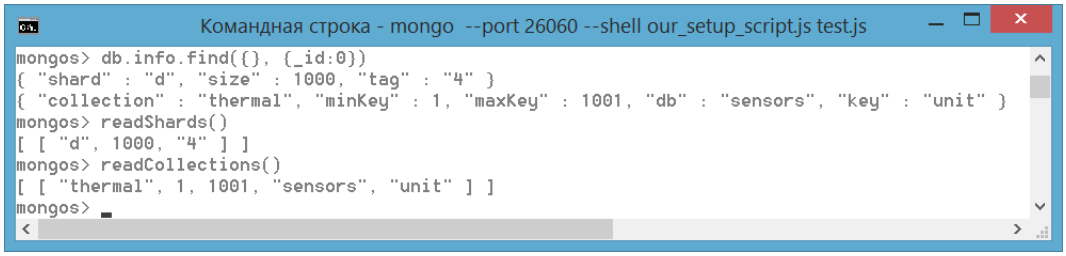

Рисунок 4.2 – Результат роботи функцій при другій ситуації

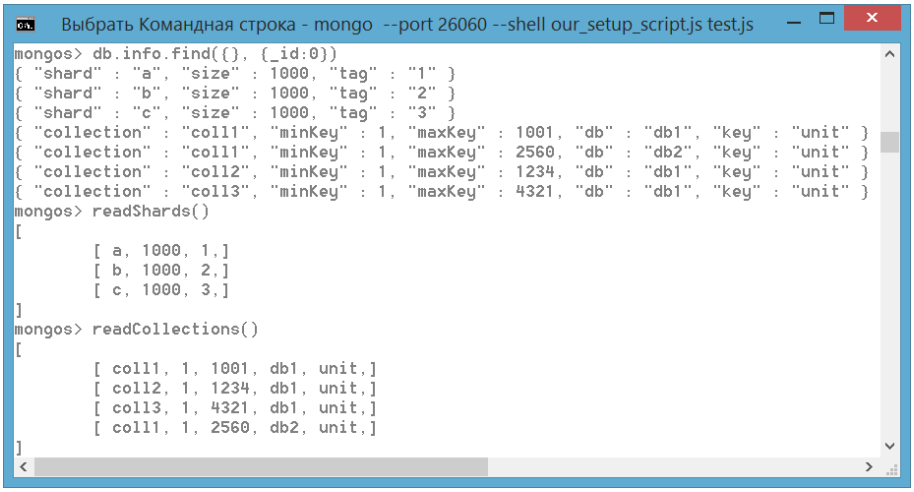

Рисунок 4.2 – Результат роботи функцій при третій ситуації

На рис. 4.1, 4.2 та 4.3 продемонстровано результати тестування роботи функцій при різних ситуаціях. З результатів стає очевидним, що програмні засоби працюють правильно в будь-якій описаній ситуації.

4.2. Тестування функцій пошуку мінімального/максимального значення поля-ключа в колекції

Тестування функцій пошуку мінімального/максимального значення ключа проводиться на ситуаціях, коли вони застосовується до:

- пустої колекції;
- колекції з даними.

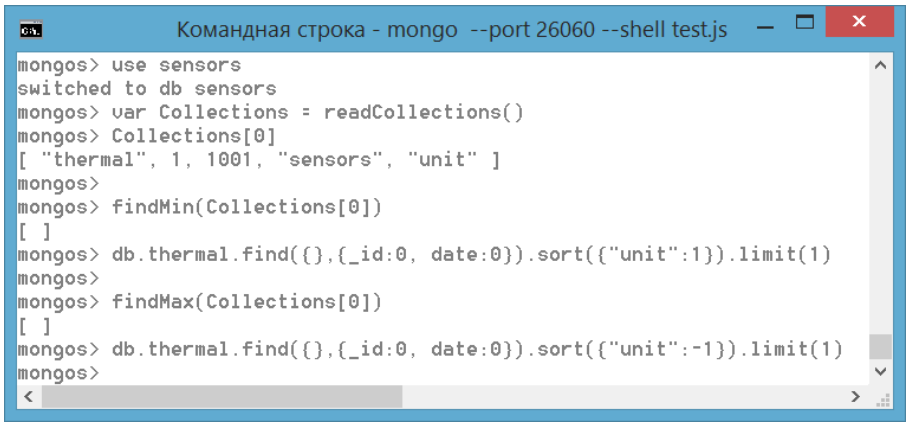

Рисунок 4.4 – Результат застосування функцій до пустої колекції

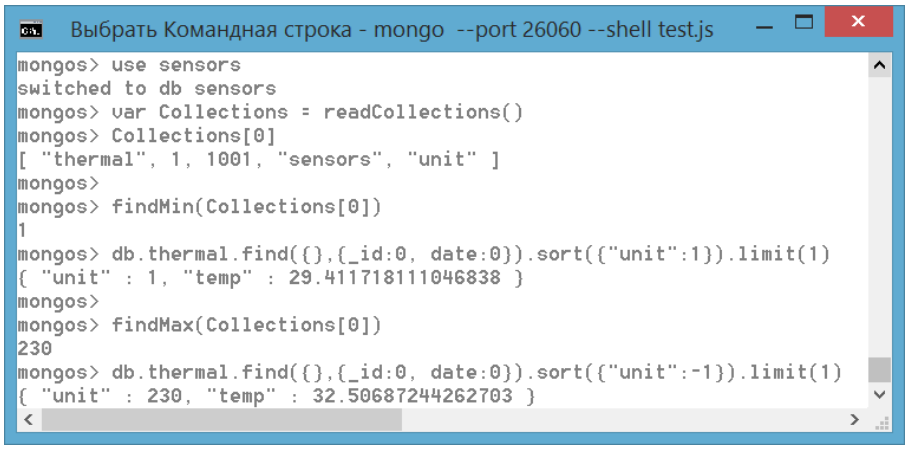

Рисунок 4.5 – Результат застосування алгоритму до колекції з даними

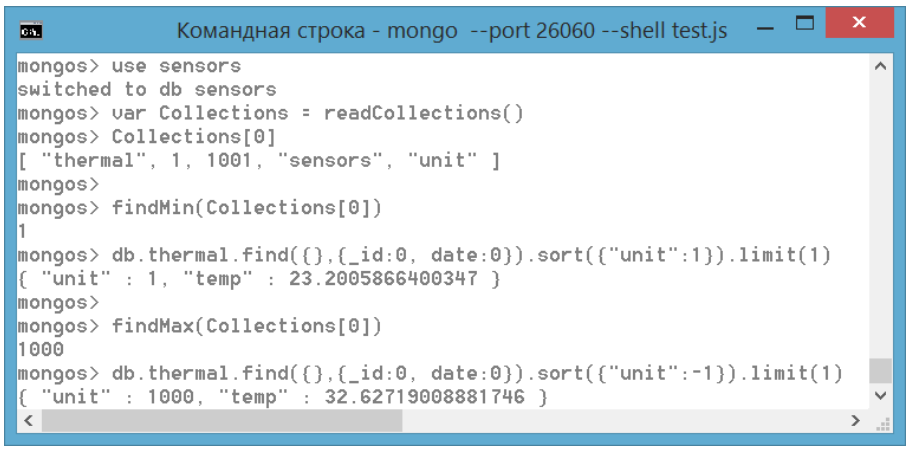

Рисунок 4.6 – Результат застосування алгоритму до колекції з даними

На рис. 4.4, 4.5 та 4.6 продемонстровано результати тестування функцій, з яких стає очевидним, що вони працюють правильно при будь-яких можливих ситуаціях.

4.3. Тестування функції перевірки чанка

Тестування функції проводиться на колекції в якій існують як звичайні чанки, так і «джамбо-чанки». Це продемонстровано на рис. 4.6 та рис. 4.7.

```
{ "unit" : { "$minKey" : 1 } } -->> { "unit" : 1 } on : b Timestamp(2, 1)
{ "unit" : 1 } -->> { "unit" : 2 } on : a Timestamp(4, 20)
{ "unit" : 2 } -->> { "unit" : 4 } on : a Timestamp(4, 21)
{ "unit" : 4 } -->> { "unit" : 5 } on : a Timestamp(4, 22)
{ "unit" : 5 } -->> { "unit" : 6 } on : a Timestamp(4, 23)
{ "unit" : 6 } -->> { "unit" : 8 } on : a Timestamp(4, 24)
{ "unit" : 8 } -->> { "unit" : 9 } on : a Timestamp(4, 25)
{ "unit" : 9 } -->> { "unit" : 10 } on : a Timestamp(4, 19)
{ "unit" : 10 } -->> { "unit" : 90 } on : a Timestamp(4, 15)
{ "unit" : 90 } -->> { "unit" : 106 } on : a Timestamp(4, 16)
{ "unit" : 106 } -->> { "unit" : 209 } on : a Timestamp(4, 6)
{ "unit" : 209 } -->> { "unit" : 251 } on : a Timestamp(4, 7)
{ "unit" : 251 } -->> { "unit" : 355 } on : b Timestamp(4, 8)
{ "unit" : 355 } -->> { "unit" : 458 } on : b Timestamp(4, 9)
{ "unit" : 458 } -->> { "unit" : 501 } on : b Timestamp(4, 10)
{ "unit" : 501 } -->> { "unit" : 605 } on : c Timestamp(4, 2)
{ "unit" : 605 } -->> { "unit" : 709 } on : c Timestamp(4, 3)
{ "unit" : 709 } -->> { "unit" : 751 } on : c Timestamp(4, 4)
{ "unit" : 751 } -->> { "unit" : 854 } on : d Timestamp(4, 11)
{ "unit" : 854 } -->> { "unit" : 959 } on : d Timestamp(4, 12)
{ "unit" : 959 } -->> { "unit" : 1001 } on : d Timestamp(4, 13)
{ "unit" : 1001 } -->> { "unit" : { "$maxKey" : 1 } } on : b Timestamp(4, 1)
```

Рисунок 4.6 – Список всіх чанків колекції

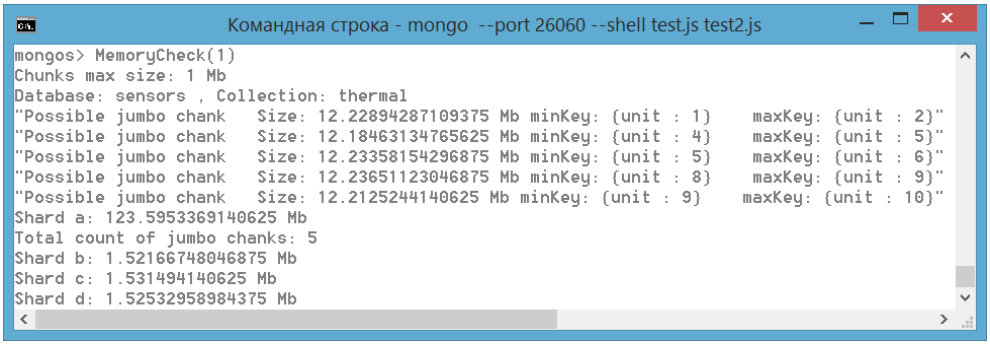

Рисунок 4.7 – Результат роботи алгоритму перевірки використаної пам’яті

На рис. 4.6 зображено, що в колекції знаходиться 22 чанки, а значення поля-ключа має діапазон від 1 до 1000 (включно). З рис. 4.7 зрозуміло, що 5 чанків у даній колекції є «джамбо-чанками».

Для тестування використовується додаткова функція «test», яка отримує інформацію про вибраний чанк для тестування та передає його дані функції перевірки. Лістинг функції test наведено нижче.

```
> function test(a , b)
> {
>       db = db.getSisterDB("config");
>       var tmp = db.chunks.find({ns: "sensors.thermal",min: {unit: a}, max:
      {unit: b}});
>       var Collections = readCollections();
>       tmp = tmp.next();
>       checkChunk(tmp, Collections[0]);
> }
```

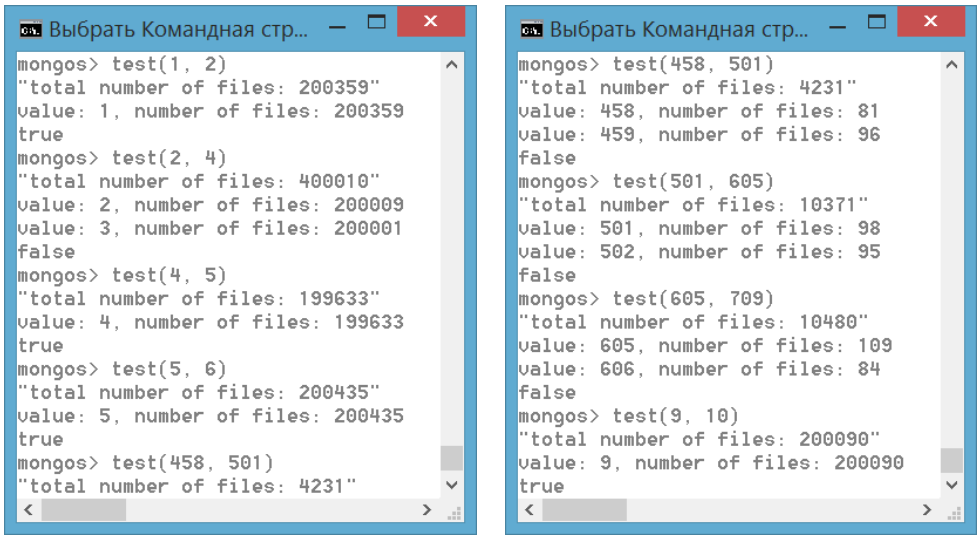

Рисунок 4.8 – Результати тестування

На рис. 4.8 наведено результати тестування з яких видно, що функція перевірки правильно визначила які з чанків містять в собі лише одне значення поля-ключа, а які більше одного. Це підтверджує правильність її роботи.

4.4. Тестування функції розрахунку відсоткового навантаження

Тестування даної функції проводиться за різної кількості шардів та в ситуаціях коли очікується різне відсоткове навантаження для серверів.

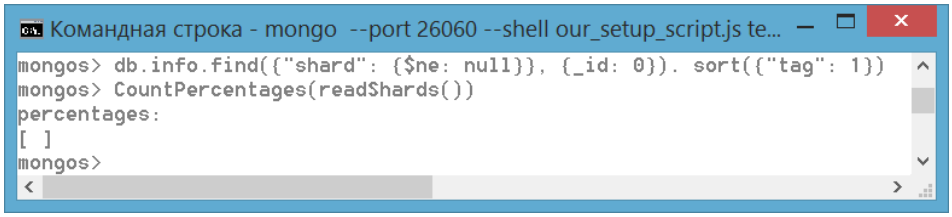

Рисунок 4.9 – Результат роботи функції за відсутності інформації про шарди

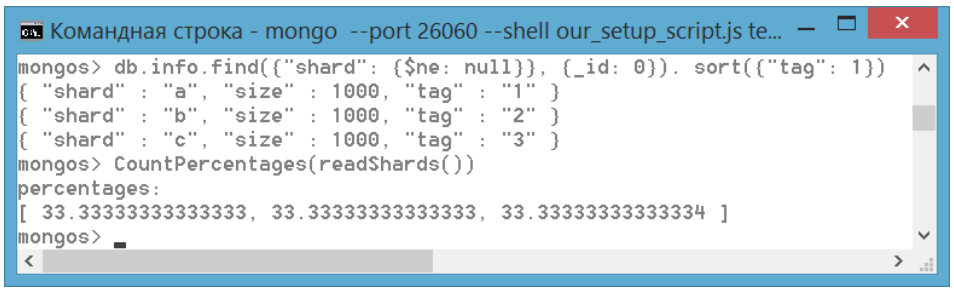

Рисунок 4.10 – Результат роботи функції при наявності інформації про три шарди, які мають однакове відсоткове навантаженням

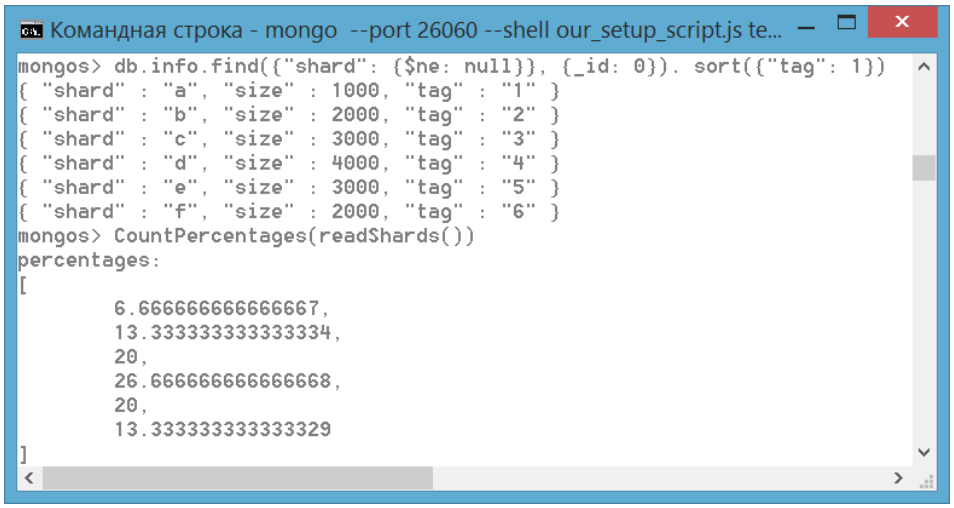

Рисунок 4.11 – Результат роботи функції при наявності інформації про шарди, які мають різне відсоткове навантаженнями

На рис. 4.9, 4.10 та 4.11 продемонстровано результати тестування роботи алгоритму в різних ситуаціях, з яких стає очевидно, що він працює правильно.

4.5. Тестування функції розрахунку кількості даних для шардів (за чанками)

Функція тестується за різної кількості шардів та при різному відсотковому навантажені, а саме:

- кількість шардів: 3 та 4
- відсоткове навантаження буде однокове для всіх шардів \ кожен наступний шард може витримати вдвічі більше навантаження ніж попередній.

Загальна кількість чанків надані функції для розподілу рівна 100.

Таблиця 4.1 - Результати тестування

Відсоткове навантаження, %	Розрахована кількість чанків, шт.
[33,33(3); 33,33(3); 33,33(3)]	[33; 33; 34]
[14,286; 28,571; 57, 143]	[14; 29; 57]
[25; 25; 25; 25]	[25; 25; 25; 25]
[6,66(6); 13,33(3); 26,66(6); 53,33(3)]	[7; 13; 27; 53]

З таблиці 4.1 можна зробити висновок, що функція працює правильно за різної кількості шардів та при різному відсотковому навантаженні.

4.6. Тестування роботи запропонованого підходу, функцій додання документів до колекцій та розрахунку кількості даних для шардів (за пам’яттю)

За реалізацію запропонованого підходу відповідають наступні функції:

- функція поділу чанків;
- функція розрахунку кількості даних для шардів (за пам’яттю);
- функція зміни границь зон;
- функція переміщення «джамбо-чанків».

Оскільки перевірка правильності роботи запропонованого підходу має відбуватись на колекціях з даними, тому під час тестування використовуються функція insertFiles, для додання документів до колекцій, та функція MemoryCheck, за допомогою якого буде проводиться перевірка правильності перерозподілу даних. Також потрібно зазначити, що запуск запропонованого підходу реалізується функцією ZoneBalancer, яка крім реалізації алгоритму зміни границь зон включає в себе й запуск трьох інших функцій, а саме: функції ChunkSpliter, функції MemoryCount та функції JumboMover.

Тестування функцій проводиться в декілька етапів:

1 етап:

- додання 1 000 000 документів до колекцій з покриттям в 100%;
- запуск програмних засобів, що реалізують запропонований підхід;
- перевірка правильності перерозподілу даних за допомогою функції перевірки використаної пам’яті.

2 етап:

- додання 2 000 000 документів до колекцій з покриттям в 10%;
- запуск програмних засобів, що реалізують запропонований підхід;
- перевірка правильності перерозподілу даних за допомогою функції перевірки використаної пам’яті .

3 етап:

- додання 5 000 000 документів до колекцій з покриттям в 60%;
- запуск програмних засобів, що реалізують запропонований підхід;
- перевірка правильності перерозподілу даних за допомогою функції перевірки використаної пам’яті .

Вище були наведені основні етапи тестування та крім них буде ще дві додаткові перевірки:

1) робота функції ZoneBalancer - у якій за розрахунок кількості чанків для кожного шарду використовується функція ChunksCount, це робиться для порівняння розподілу даних за кількістю чанків з розподілом за пам’яттю;
2) робота функції ZoneBalancer у ситуації коли у кожного шарду різне відсоткове навантаження.

Тестування роботи функцій проводиться на двох колекціях – колекції «thermal» та колекції «gps». Ключем шардингу для колекції «thermal» буде поле «unit», а для колекції «gps» - «Unit».

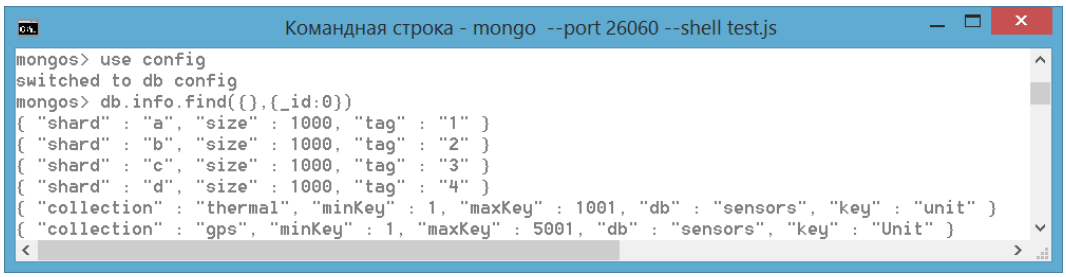

Рисунок 4.12 – Інформація про колекції та шарди

На рис. 4.12 зображено налаштування які зберігаються в колекції «info» бази даних «config». Дані налаштування будуть використовуватись на трьох етапах тестування.

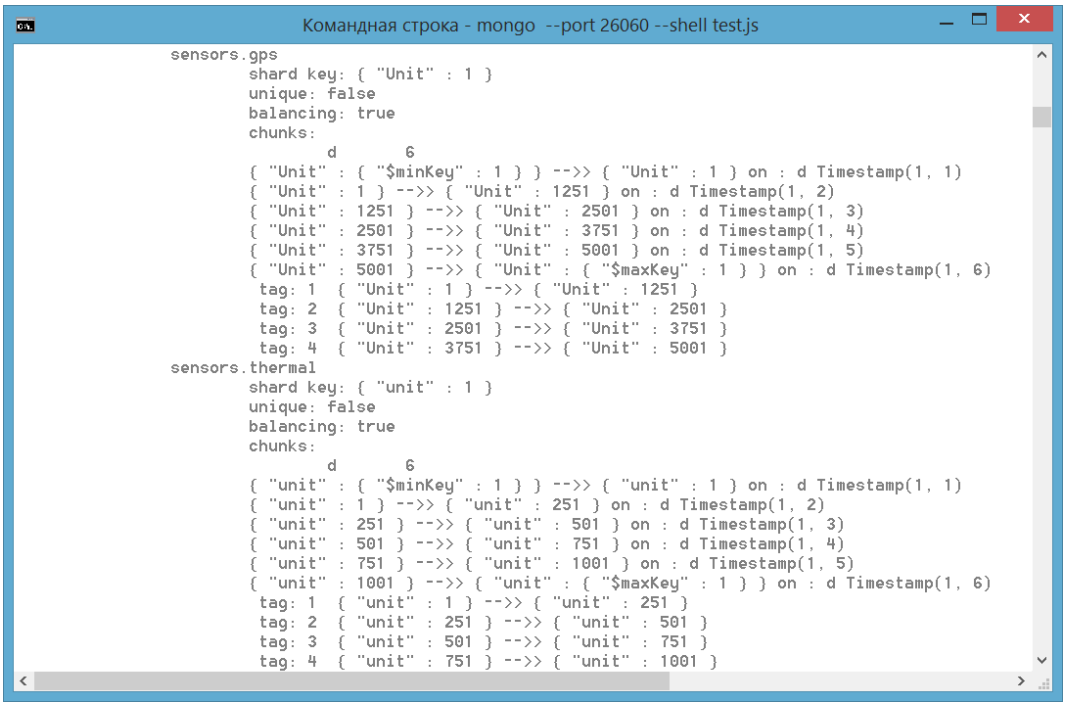

Рисунок 4.13 – Початкові налаштування колекцій

На рис. 4.13 зображено початкові налаштування розподілу колекції з яких зрозуміло, що:

- для кожної колекції існує 4 зони, вказані їх границі;
- у обох колекцій включений процес балансування;
- значення поля-ключа не є унікальним;
- у кожній колекції знаходиться 6 чанків (вони пусті – рис. 4.13).

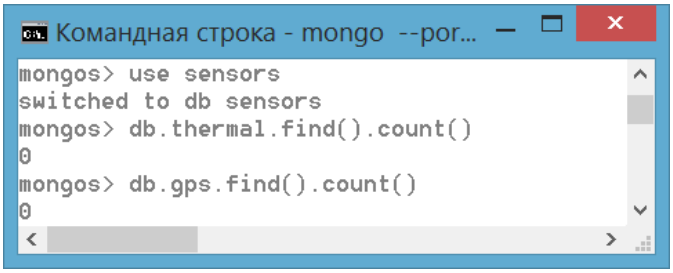

Рисунок 4.14 – Початковий стан колекцій «thermal» та «gps»

На рис. 4.14 зображено, що колекції є пустими, оскільки кількість документів в них дорівнює 0.

4.6.1. Перший етап тестування

Даний етап тестування починається з додання 1 000 000 документів до колекцій з 100% покриттям виставлених діапазонів значень полів-ключів.

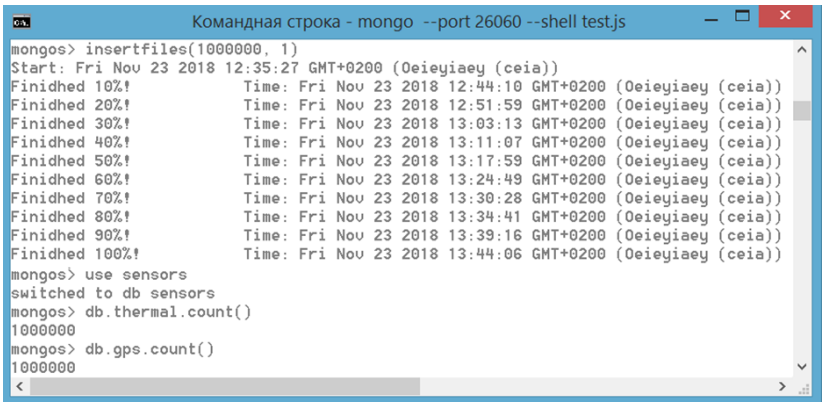

Рисунок 4.15 – Додання 1 000 000 документів з покриттям 100%

На рис 4.15 продемонстровано роботу функції insertFiles та її результат. До обох колекцій було добавлено по 1 000 000 документів. Після додання документів стан колекцій змінився, для кожного шарду були створені нові чанки з даними (інформація про кількість чанків та розмір використаної пам’яті у кожному шарді надана в таблиці 4.2).

Таблиця 4.2 - Стан шардингу колекцій з 1 000 000 документів

	Шард	К-сть чанків, шт.	Границі зон	Використана пам’ять, Мб
Колекція «thermal»	a	37	[1 ; 251)	15,215
	b	37	[251 ; 501)	15,277
	c	37	[501 ; 751)	15,284
	d	39	[751 ; 1001)	15,259
Колекція «gps»	a	31	[1 ; 1251)	21,193
	b	31	[1251 ; 2501)	21,253
	c	31	[2501 ; 3751)	21,216
	d	33	[3751 ; 5001)	21,215

Вказані у таблиці 4.2 розміри використаної пам’яті знаходились за допомогою функції MemoryCheck. Крім цього, за її ж допомогою було виконано перевірку колекцій на наявність «джамбо-чанків» - на даному етапі тестування вони відсутні, оскільки даних недостатньо для їх появи.

Далі відбувся запуск функції ZoneBalancer (повний лістинг результату якої надано в додатку 2).

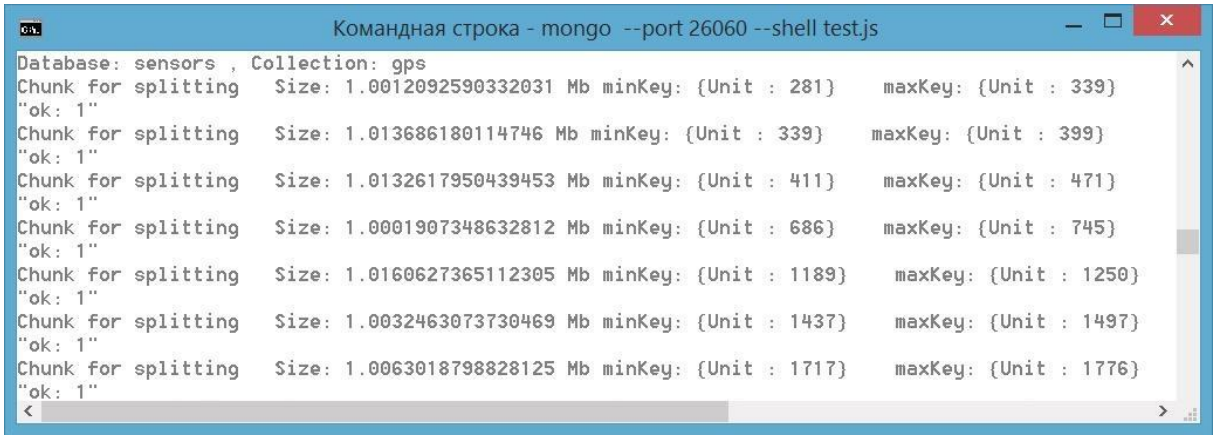

Рисунок 4.16 – Виконання функції ChunkSpliter для колекції «gps»

На рис. 4.16 продемонстровано роботу функції ChunkSpliter для колекції «gps» з якої стає очевидним, що функція успішно знаходить чанки розмір яких перевищує максимально допустимий (1 Мб) та поділяє їх. Це означає, що кількість чанків у колекції змінилась порівняно з даними в таблиці 4.2.

Таблиця 4.3 - Результати розрахунків під час виконання запропонованого підходу на 1 етапі тестування

	Шард	К-сть чанків, шт.	Розраховані границі зон	Розрахована пам’ять, Мб
Колекція «thermal»	a	38	[1 ; 251)	15,215
	b	37	[251 ; 501)	15,277
	c	37	[501 ; 751)	15,284
	d	38	[751 ; 1001)	15,259
Колекція «gps»	a	37	[1 ; 1251)	21,193
	b	34	[1251 ; 2500)	21,233
	c	37	[2500 ; 3750)	21,218
	d	37	[3750 ; 5001)	21,214

У таблиці 4.3 наведено результат роботи запропонованого підходу. Під час його виконання відбувся поділ чанків (рис. 4.16), перерозрахунок кількості чанків для кожного шарду (за пам’яттю) та зміна деяких границь зон (у колекції «gps» границі зон для шардів «b», «c» та «d» змінились). З результатів можна сказати, що при доданні документів до колекцій зі 100% покриттям та відсутності «джамбо-чанків» запропонований підхід не вніс суттєвих змін в розподіл даних.

Оскільки на даному етапі тестування відсутні «джамбо-чанки», то функція JumboMover не виконала жодного переміщення, але тимчасова зміна максимально допустимого розміру чанків відбулась. (рис. 4.17).

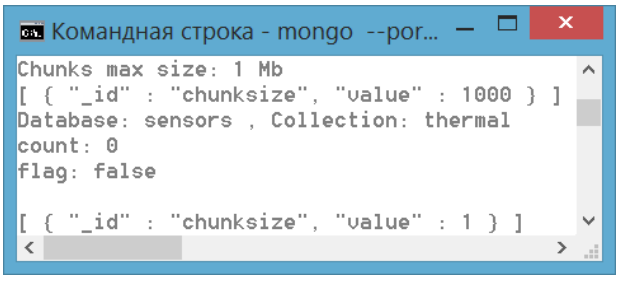

Рисунок 4.17 – Виконання функції переміщення «джамбо-чанків» для колекції «thermal»

Після завершення балансувальником переміщень чанків на відповідні їм шарди відбулась перевірки використаної пам’яті за

допомогою функції MemoryCheck, яка підтвердила правильність розрахунків.

4.6.2. Другий етап тестування

Стан колекцій на початку другого етапу відповідає їх стану в кінці попереднього етапу, тобто інформації наданій в таблиці 4.3.

Даний етап тестування починається з додання 2 000 000 документів до колекцій з 10% покриттям виставлених діапазонів значень полів-ключів. Тобто після додання записів в обох колекціях буде знаходиться по 3 000 000 документів.

Після додання документів стан колекцій змінився. Для кожного шарду були створені нові чанки з даними (інформація про кількість чанків та розмір використаної пам’яті у кожному шарді надана в таблиці 4.4).

Таблиця 4.4 - Стан шардингу колекцій з 3 000 000 документів

	Шард	К-сть чанків, шт.	Границі зон	Використана пам’ять, Мб	Jumbo chunks
Колекція «thermal»	a	96	[1 ; 251)	137,285	42
	b	37	[251 ; 501)	15,277	0
	c	37	[501 ; 751)	15,284	0
	d	38	[751 ; 1001)	15,259	0
Колекція «gps»	a	327	[1 ; 1251)	190,947	0
	b	34	[1251 ; 2500)	21,233	0
	c	37	[2500 ; 3750)	21,218	0
	d	39	[3750 ; 5001)	21,231	0

Вказані у таблиці 4.4 розміри використаної пам’яті та кількість «джамбо-чанків» знаходились за допомогою функції перевірки використаної пам’яті (лістинг її роботи надано в додатку 3). З перевірки видно, що вже на даному етапі було виявлено 42 «джамбо-чанків» в колекції «thermal».

Далі відбувся запуск функції ZoneBalancer (повний лістинг результату якої надано в додатку 4). Під час її роботи відбувся запуск функції поділу чанків, яка відпрацювала так само як і на попередньому

етапі тестування – це означає, що кількість чанків у колекції знову змінилась порівняно з даними в таблиці 4.4.

Таблиця 4.5 - Результати розрахунків під час виконання запропонованого підходу на 2 етапі тестування

	Шард	К-сть чанків, шт.	Розраховані границі зон	Розрахована пам’ять, Мб	Jumbo chunks
Колекція «thermal»	a	37	[1 ; 37)	46,033	36
	b	36	[37 ; 73)	46,24	36
	c	53	[73 ; 260)	45,56	28
	d	111	[260 ; 1001)	45,273	0
Колекція «gps»	a	125	[1 ; 179)	63,509	0
	b	114	[179 ; 357)	63,5	0
	c	118	[357 ; 1219)	63,411	0
	d	109	[1219 ; 5001)	63,684	0

У таблиці 4.5 наведено результат роботи запропонованого підходу. Під час його виконання відбувся поділ чанків, перерозрахунок кількості чанків для кожного шарду (за пам’яттю), зміна границь зон та переміщення «джамбо-чанків» (рис. 4.18). Кількість «джамбо-чанків» знаходилась в подальшому при перевірці використаної пам’яті функцією MemoryCheck (лістинг її роботи надано в додатку 5). З даних результатів можна сказати, що:

- після поділу чанків були виявлені «нові» «джамбо-чанки», причиною їх появи є те, що до поділу вони були у складі чанку який містив у собі декілька значень поля-ключа (це суперечило поняттю «джамбо-чанк»), а після поділу вони стали окремими чанками;
- для рівномірного розподілу даних кількість чанків на кожному шарді має бути різною, чого неможливо досягти за допомогою балансувальника без використання зон – це підтверджує актуальність розроблення запропонованого підходу;
- у колекції «thermal» уже наявні «джамбо-чанки, тобто з цього моменту досягти рівномірного розподілу даних за допомогою

стандартного підходу розподілу (за чанками) буде майже неможливо.

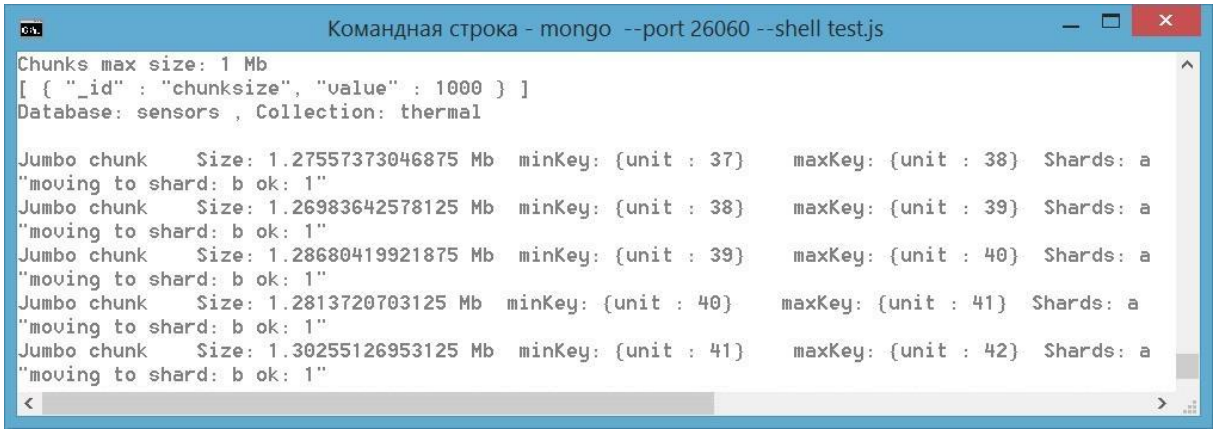

Рисунок 4.18 – Виконання функції переміщення «джамбо-чанків»

На рис 4.18 зображено виконання функції JumboMover, оскільки розмір «джамбо-чанків» не перевищує 1.3 Мб, то дані чанки могли бути переміщенні балансувальником (див. пункт 2.1.5), але на відміну від балансувальника дана функція може переміщувати чанки з більшими розмірами (див. рис. 4.20).

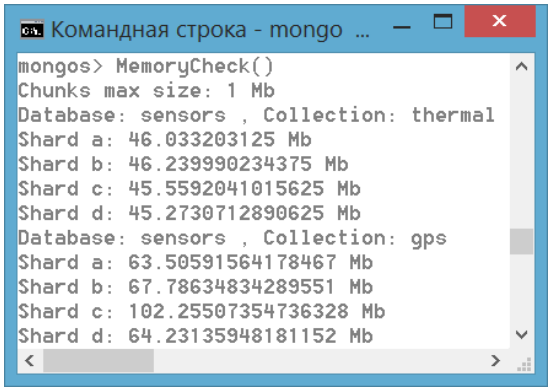

Рисунок 4.19 – Неправильний результат роботи функції перевірки використаної пам’яті

Також потрібно зазначити, що перед тим як був отриманий правильний результат роботи функції MemoryCheck (додаток 5), було проведено додаткові запуски функції перевірки використаної пам’яті, які виводили результати невідповідні до розрахованих під час роботи запропонованого підходу (таблиця 4.5), приклад такого результату продемонстровано на рис. 4.19. Неправильний вивід був отриманий через те, що під час виконання функції MemoryCheck, база даних працювала зі

збереженими даними в колекціях для коректування метаданих (обновлення вказівників на дані) збережених на конфігураційних серверах.

4.6.3. Третій етап тестування

Стан колекцій на початку третього етапу відповідає їх стану в кінці другого етапу, тобто інформації наданій в таблиці 4.5.

Третій етап тестування починається з додання 5 000 000 документів до колекцій з 60% покриттям виставлених діапазонів значень полів-ключів, тобто після додання записів в обох колекціях буде знаходиться по 8 000 000 документів.

Після додання документів стан колекцій змінився, а саме для кожного шарду були створені нові чанки з даними (інформація про кількість чанків та розмір використаної пам’яті у кожному шарді надана в таблиці 4.6).

Таблиця 4.6 - Стан шардингу колекцій з 8 000 000 документів

	Шард	К-сть чанків, шт.	Границі зон	Використана пам’ять, Мб	Jumbo chunks
Колекція «thermal»	a	37	[1 ; 37)	64,311	36
	b	36	[37 ; 73)	64,585	36
	c	143	[73 ; 260)	140,72	28
	d	309	[260; 1001)	218,665	0
Колекція «gps»	a	125	[1 ; 179)	88,694	0
	b	114	[179 ; 357)	88,671	0
	c	298	[357 ; 1219)	185,339	0
	d	568	[1219 ; 5001)	316,313	0

Вказані у таблиці 4.6 розміри використаної пам’яті та кількість «джамбо-чанків» знаходились за допомогою функції перевірки використаної пам’яті (лістинг її роботи надано в додатку 6). З результатів стає очевидним, що розподіл нерівномірний.

Далі відбувся запуск функції ZoneBalancer (повний лістинг результату якої надано в додатку 7). Під час її роботи відбувся запуск функції поділу чанків, яка відпрацювала так само як і на попередньому

етапі тестування – це означає, що кількість чанків у колекції знову змінилась порівняно з даними в таблиці 4.6.

Таблиця 4.7 - Результати розрахунків під час виконання запропонованого підходу на 3 етапі тестування

	Шард	К-сть чанків, шт.	Розраховані границі зон	Розрахована пам’ять, Мб	Jumbo chunks
Колекція «thermal»	a	69	[1 ; 69)	123,509	68
	b	144	[69 ; 213)	122,813	38
	c	216	[213 ; 429)	123,154	3
	d	234	[429 ; 1001)	122,336	0
Колекція «gps»	a	271	[1 ; 343)	170,881	1
	b	284	[343 ; 1067)	170,728	1
	c	295	[1067 ; 2140)	170,783	0
	d	318	[2140 ; 5001)	170,734	0

У таблиці 4.7 наведено результат роботи запропонованого підходу. Під час його виконання відбувся поділ чанків, перерозрахунок кількості чанків для кожного шарду (за пам’яттю), зміна границь зон та переміщення «джамбо-чанків». Кількість «джамбо-чанків» знаходилась в подальшому при перевірці використаної пам’яті функцією MemoryCheck (лістинг її роботи надано в додатку 8). З даних результатів можна сказати, що:

- для рівномірного розподілу даних кількість чанків на кожному шарді має бути різною, чого неможливо досягти за допомогою балансувальника без використання зон – це підтверджує актуальність розроблення запропонованого підходу;
- у колекцій уже наявні «джамбо-чанки», тобто з цього моменту досягти рівномірного розподілу даних за допомогою стандартного підходу розподілу (за чанками) буде майже неможливо.

Робота функції JumboMover (див. додаток 7) нічим не відрізнялась від його роботи на другом етапі тестування, хоча розміри «джамбо-чанків» в середньому становили 1.8 Мб – це пов’язано з тим, що до початку їх

переміщення налаштування зі значенням максимально допустимого розміру успішно було змінено (на відміну від його роботи підчас додаткового тестування – див. рис. 4.20).

4.6.4. Результати додаткових тестів

У даному підрозділі розглядаються результати перевірки роботи запропонованого підходу при розподілі за кількістю чанків та перевірка розподілу за різним відсотковим навантаженням.

4.6.4.1. Результат роботи при розподілі за кількістю чанків

Для цього тесту налаштування залишились тими ж, сам же тест проводиться після завершення третього етапу тестування.

У таблиці 4.8 надані розрахунки роботи запропонованого підходу при розподілі за чанками, а повний результат роботи функції ZoneBalancer для даного тесту наданий в додатку 9.

Таблиця 4.8 - Результати розрахунків під час виконання запропонованого підходу (розподіл за чанками)

	Шард	К-сть чанків, шт.	Розраховані границі зон	Використана пам’ять, Мб	Jumbo chunks
Колекція «thermal»	a	166	[1 ; 166)	219,599	103
	b	166	[166 ; 332)	94,596	0
	c	166	[332 ; 498)	94,555	0
	d	165	[498 ; 1001)	83,061	0
Колекція «gps»	a	292	[1 ; 369)	186,344	2
	b	292	[369 ; 1164)	170,606	0
	c	292	[1164 ; 2228)	168,707	0
	d	292	[2228 ; 5001)	156,348	0

У таблиці 4.8 наведено результат роботи запропонованого підходу. Під час його виконання відбувся поділ чанків, перерозрахунок кількості чанків для кожного шарду (за чанками), зміна границь зон та переміщення «джамбо-чанків». Кількість використаної пам’яті та кількість «джамбо-

чанків» знаходилась в подальшому при використанні функції MemoryCheck (лістинг її роботи надано в додатку 10).

Отримані результати в таблиці 4.8 знову підтверджує, що розподіл за кількістю чанків не завжди буде рівномірним.

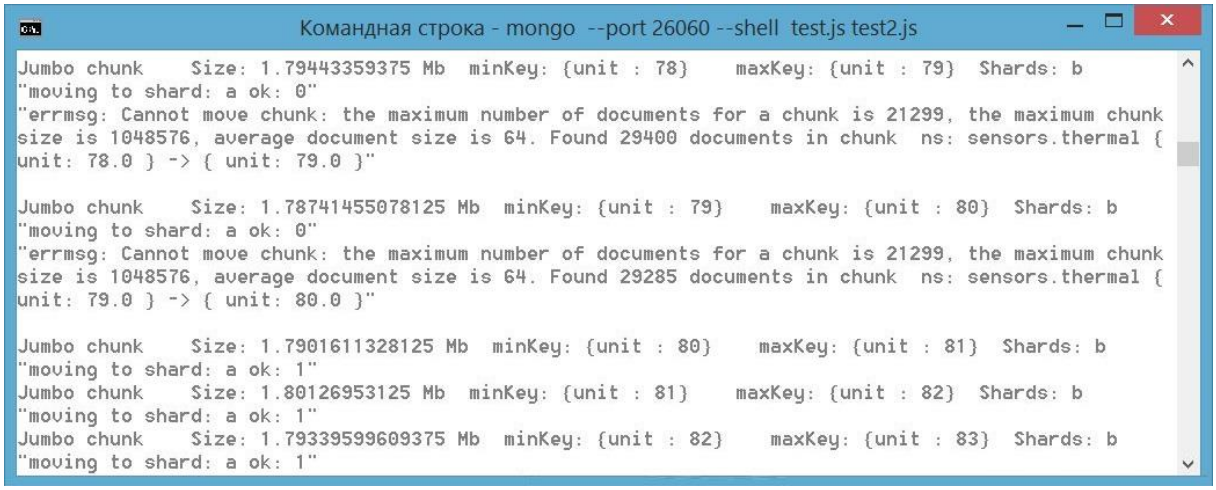

Рисунок 4.20 – Робота функції переміщення «джамбо-чанків»

На рис. 4.20 наведено роботу функції JumboMover для даного тесту. На відміну від його попередніх дій на другому та третьому етапі тестування, в цьому випадку значення максимально допустимого розміру чанків не встигли змінитись повністю до спроби переміщення «джамбо-чанків» (в деяких частинах системи воно залишилось рівним 1 Мб.), тому операції спочатку були неуспішними.

4.6.4.2. Результат тесту розподілу за різним відсотковим навантаженням

Для даного тесту були змінені налаштування розподілу (рис. 4.21). Основною метою даного тесту є перевірка можливості його застосування при різних відсоткових навантаженнях.

Тестування проводиться на колекції «thermal» після того як в неї було добавлено 2 000 000 документів з покриттям 100%.

Повний результат роботи функції ZoneBalancer для даного тесту наданий в додатку 11.

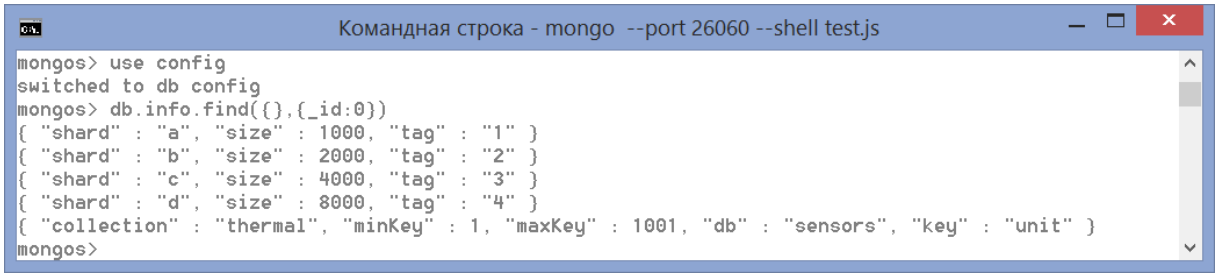

Рисунок 4.21 – Налаштування збережені в колекції «info» бази даних «config»

Таблиця 4.9 - Результат перевірки додаткового тесту

Шард	Відсоткове навантаження	К-сть чанків, шт.	Розраховані границі зон	Розрахована пам’ять, Мб
a	6,66(6)	13	[1 ; 50)	8,4375
b	13,33(3)	26	[50 ; 148)	17,6886
c	26,66(6)	69	[148 ; 430)	34,4119
d	53,33(3)	139	[430 ; 1001)	69,0265

З таблиці 4.9 стає зрозумілим, що розроблені програмні засоби надають можливість встановлювати відсоткове навантаження для кожного шарду окремо.

4.6.5. Результати тестування

З отриманих результатів тестування стає очевидно, що всі програмні засоби, які приймали участь в тестуванні, працюють правильно. Також знову було доведено, що розподіл за чанками не завжди є рівномірним.

4.7. Тестування функцій створення та видалення зон

Тестування функції створення зон проводиться на наступних ситуаціях:

- мінімальний ключ є від’ємним числом, а максимальний – додатнім (ситуація №1);
- різниця між максимальним та мінімальним ключем менша кількості шардів (ситуація №2);

- різниця між максимальним та мінімальним ключем ненабагато більша кількості шардів (ситуація №3);
- різниця між максимальним та мінімальним ключем набагато більша кількості шардів (ситуація №4);
- створення зон для декількох колекцій (ситуація №5).

Крім цього тести потрібно робити при таких налаштуваннях:

- різна кількість вказаних шардів (наприклад, 3 та 4);
- у шардів однакове відсоткове навантаження;
- у шардів різне відсоткове навантаження;

Тестування функції видалення проводиться після кожної етапу тестування функції створення зон. Крім цього потрібно провести її тестування при ситуації, коли крім зон для вказаних колекцій існують і зони для інших колекцій.

Результати надані у таблиці 4.10, де «+» - тест успішний, а «-» - неуспішний. Лістинг тестів надано в додатку 12.

Таблиця 4.10 - Результати тестувань

ситуація	Функція створення зон				Функція видалення зон	
	3 шарди		4 шарди		3 шарди	4 шарди
	Відсоткове навантаження:					
	однакове	різне	однакове	різне		
№1	+	+	+	+	+	+
№2	+	+	+	+	+	+
№3	+	+	+	+	+	+
№4	+	+	+	+	+	+
№5	+	+	+	+	+	+

З результатів наведених в таблиці 4.10 видно, що функції працюють правильно у всіх вибраних ситуаціях.

Потрібно зазначити, що додатковий тест функції видалення зон також успішний.

ВИСНОВКИ

Отже, було запропоновано та реалізовано новий підхід розподілу в базі даних MongoDB. Даний підхід розподіляє дані більш рівномірно порівняно з існуючими підходами, крім цього для нього була реалізована можливість встановлення відсоткового налаштування для кожного серверу.

Однією з основних причин розробки нового підходу стала відсутність методів розподілу даних в кластері з урахуванням «джамбо-чанків». Під час тестування існуючих підходів було виявлено їх спільну проблему – при наявності джамбо-чанків стандартний розподіл за кількістю чанків стає неточним, кількість використаної пам’яті на кожному шарді в кластері була суттєво різною. Тому в запропонований підхід було включено додаткові процеси обробки «джамбо-чанків», які зменшують їх негативний вплив як на розподіл даних так і на роботу системи в цілому.

Було проведено тестування розроблених програмних засобів, як і тих, що реалізують запропонований підхід, так і додаткових. Результатами тестування було підтверджено, що всі розроблені функції працюють правильно.

Також було виявлено, що запропонований підхід не вносить суттєвих змін в ситуаціях схожих з першим етапом тестування (див. пункт 4.6.1). Але під час другого та третього етапів він вніс зміни в налаштуваннях без яких досягти рівномірного розподілу було б неможливо – це підтверджує його необхідність в таких ситуаціях.

Якщо ж розглядати розроблені програмні засоби з точки зору їх практичної цінності, то потрібно зазначити, що вони можуть бути використані для реалізації автоматизованого розподілу даних з заданими налаштуваннями. Але є одна рекомендація щодо їх використання: використовувати запропонований підхід бажано лише при досягнені деяких критеріїв, як і у процесі балансування (див. підрозділ 2.1), бо

замість підвищення продуктивності роботи системи можна її перенавантажити.

На даний момент, розроблені програмні засоби працюють лише коли у якості ключа шардингу виступає одне поле, тому у подальшому було б доцільно реалізувати можливість роботи зі складеними ключами шардингу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Прамодкумар Дж. Садападж, Мартин Фаулер. NoSQL. Новая методология разработки нереляционных баз данных. Москва: издательство “Вильямс”, 2013.
2. NOSQL DATABASES [Електронний ресурс] – 2011 – Режим доступу: http://nosql-database.org/ – Дата доступу: грудень 2018.
3. Ranged Sharding – [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/core/ranged-sharding/index.html – Дата доступу: грудень 2018.
4. Kristina Chodorow. MongoDB: The Definitive Guide, 3rd Edition / Kristina Chodorow, Shannon Bradshaw // O'Reilly Media, Inc. – 2018.
5. Hashed Sharding– [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/core/hashed-sharding/index.html – Дата доступу: грудень 2018.
6. Zones – [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/core/zone-sharding/ – Дата доступу: грудень 2018.
7. mongo Shell Methods – [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/reference/method/index.html – Дата доступу: грудень 2018.
8. Shard Keys – [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/core/sharding-shard-key/#shard-key-range – Дата доступу: грудень 2018.
9. Sharded Cluster Balancer – [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/core/sharding-balancer-administration/#cluster-balancer – Дата доступу: грудень 2018.
10. Glossary – [Електронний ресурс] – 2018 – Режим доступу: https://docs.mongodb.com/manual/reference/glossary – Дата доступу: грудень 2018.

11. Большие Данные - новая теория и практика [Електронний ресурс] – 2011 – Режим доступу: https://www.osp.ru/os/2011/10/13010990/ – Дата доступу: грудень 2018.
12. Dean J., Ghemawat S. Mapreduce: simplified data processing on large clusters // OSDI’04: Proceedings of the 6th conference on symposium on operation systems design and implementation. USENIX Association. 2004.
13. Буторин Д.Н. Разработка баз данных в MongoDB. Красноярск: КГПУ, 2013.
14. Кайл Бэнкер. MongoDB в действии. Москва: издательство “ДМК Пресс”, 2012.